Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

Sebastian Müller

Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

**Sebastian Müller**

# Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Franz Korf
Zweitgutachter : Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 06.Oktober 2014

**Autor**
**Sebastian Müller**

**Thema der Arbeit**
Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways

**Stichworte**
Gateway, Time-Triggered Ethernet, CAN, FlexRay, Realtime, Simulation, OMNeT++

**Kurzzusammenfassung**
Um zukünftige Anforderungen an Flexibilität, Skalierbarkeit und Bandbreite in Kommunikationsnetzen der Automobil-Industrie sowie der Luft- und Raumfahrt zu gewährleisten, werden neue Technologien benötigt. Time-Triggered Ethernet (TTEthernet) der TTTech Computertechnik AG kann als Backbone-Netzwerk in Automobilen eingesetzt werden. Um die Anbindung herkömmlicher Kommunikationssysteme wie CAN und FlexRay zu ermöglichen, werden Gateways benötigt. In dieser Arbeit wird ein Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways entwickelt, das CAN-Bussysteme an ein TTEthernet Backbone-Netzwerk anbindet. Die Entwicklung eines dynamischen Ansatzes für die Nachrichten-Akkumulation, ermöglicht ein adaptives Verhalten des Gateways. Eine Erweiterung um weitere Bussysteme ist ohne weiteres möglich.

**Author**
**Sebastian Müller**

**Title of the Paper**
Simulation model of a Multi-Bus Realtime Ethernet Gateway

**Keywords**
Gateway, Time-Triggered Ethernet, CAN, FlexRay, Realtime, Simulation, OMNeT++

**Abstract**
In order to meet future requirements in terms of flexibility, scalability and bandwidth of communication networks in the automotive- and aerospace-context, new technologies are needed. Time-Triggered Ethernet (TTEthernet) developed by TTTech Computertechnik AG could be used as a backbone-network within a car. To enable the connection of conventional communication systems like CAN and FlexRay, gateways are required. This paper develops a simulation model of a multi-bus realtime Ethernet gateway, which connects CAN-bussystems to a TTEthernet backbone-network. The development of a dynamic approach of message accumulation allows for the adaptive functioning of a gateway. An extension of further bussystems is prepared.

# Danksagung

Danken möchte ich in erster Linie meinem Betreuer, Herrn Prof. Dr. -Ing. Korf, der stets mit konstruktiver Kritik behilflich gewesen ist. Dank seiner hervorragenden Fachkenntnisse konnten meine Gedanken immer wieder auf die richtige Spur gelenkt werden. Auch möchte ich allen Mitgliedern der CoRE Research Group[1] danken, die bei technischen Fragestellungen immer ein offenes Ohr hatten.

Meiner Tante, Dr. Babara Jeanrenaud, möchte ich herzlich dafür danken, dass sie als Fachfremde eine Korrektur-Lesung der Arbeit durchgeführt hat. Dadurch konnte eine wesentliche Qualitätssteigerung in der sprachlichen Ausdruckweise erreicht werden.

Nicht zuletzt möchte ich meinen Eltern für die finanzielle Unterstützung und Motivation während des Studiums danken!

[1] (CoRE Research Group)

# Inhaltsverzeichnis

# 1 Einleitung

Ziel der Arbeit ist es, ein Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways zu entwickeln, sodass CAN[2] und Time-Triggered Ethernet[6] in einer Backbone[3]-Architektur analysiert und getestet werden können. Auf diese Weise können realitätsnahe Backbone-Architekturen mit verschiedenen Software-Lösungen simuliert werden.

Moderne Automobile haben eine Vielzahl an elektronischen Elementen, die sowohl für Fahrsicherheit als auch Komfort und Entertainment im Fahrzeug verantwortlich sind. Steuergeräte, Sensorik und Kommunikationssysteme bilden die Basis für diese Funktionen und erstrecken sich über das gesamte Fahrzeug. Aufwändige Assistenz-Systeme, wie z.B. Fußgänger-Notbremsassistenten, werden häufig simultan durch unterschiedliche Sensor-Daten versorgt, um eine Verifikation des erkannten Sachverhaltes durchführen zu können. Sensoren wie LiDAR[4] und Kamera generieren große Datenmengen, was zu einer starken Auslastung der Kommunikationssysteme führen kann. Gängige Kommunikationssysteme im Automobil sind Bussysteme wie CAN und FlexRay, wobei FlexRay mit einer maximalen Übertragungsrate von 10 Mbit/s schneller ist als CAN mit maximal 1 Mbit/s. Diese Bussysteme erfüllen unterschiedliche Echtzeit-Anforderungen und werden für verschiedene Anwendungen eingesetzt. Unter Echtzeit-Anforderungen sind Zeitvorgaben zu verstehen, die durch das Bussystem sichergestellt werden müssen. Die hier genannten Bussysteme können allerdings den wachsenden Anforderungen der Automobil-Industrie nicht genügen, da insbesondere eine größere Bandbreite und domänenübergreifende[5] Interaktion der Kommunikationssysteme benötigt wird.

TTTech Computertechnik AG[6] hat auf Basis der Ethernet Technologie ein neues Netzwerk entwickelt, das die unterschiedlichen Echtzeit-Anforderungen der bekannten Bussysteme in einem Netzwerk vereint und darüber hinaus hervorragende Eigenschaften in Bezug auf Skalierbarkeit und Bandbreite bietet. Es trägt den Namen Time-Triggered Ethernet (TTEthernet) und wurde bereits 2011 durch SAE[7] standardisiert. Dieses Netzwerk ist speziell für die Fahrzeug- und Flugzeugindustrie entwickelt worden und könnte als alleiniges Netzwerk im Automobil alle Anforderungen abdecken. Die Interaktion zwischen den

[2] CAN (Robert Bosch GmbH, 1991)
[3] Verbindender Kernbereich eines Kommunikationsnetzes
[4] LiDAR (Light detection and ranging)
[5] Über mehrere Bereiche hinweg
[6] (TTTech Computertechnik AG, 2008)
[7] SEA (Society of Automotive Engineers (SAE International Group, 2011))

verschiedenen Domänen eines Automobil-Netzwerkes wäre damit unproblematisch. Um eine schrittweise Integration des Time-Triggered Ethernet zu ermöglichen, wird ein Backbone-Netzwerk mit der neuen TTEthernet Technologie in das Automobil integriert. Herkömmliche Bussysteme werden über Gateways an das Backbone-Netzwerk angebunden und können so untereinander kommunizieren. Kostengünstige Bussysteme in hochspezialisierten Anwendungsgebieten wie Motorsteuerung können dadurch weiterhin unverändert genutzt werden. Hohe Investitionssummen für neue Steuergeräte-Software bleiben den Herstellern damit erspart, während Raum für neue, domänenübergreifende Anwendungen geschaffen wird.

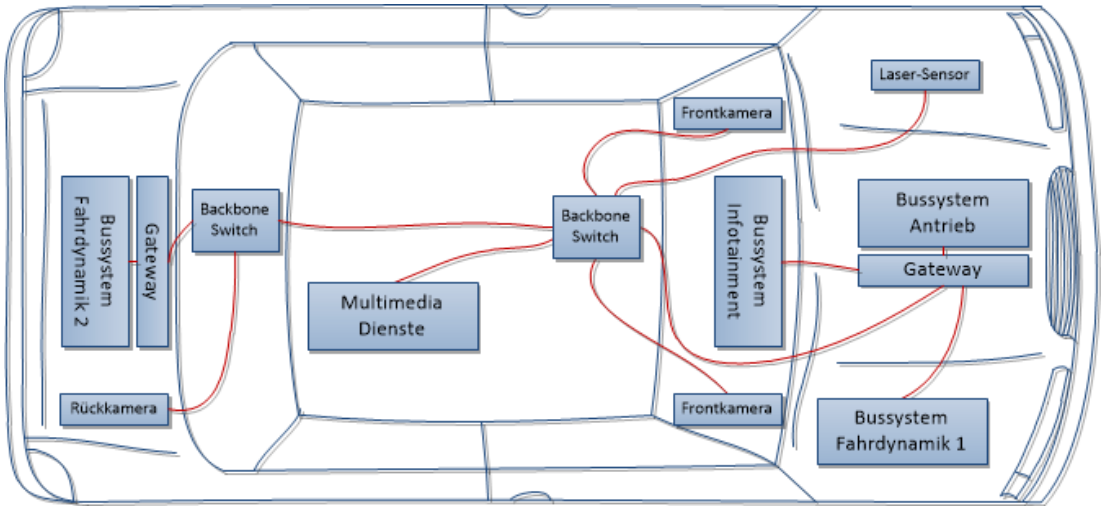


**Abbildung I: Backbone-Architektur**

In Abbildung I ist exemplarisch eine Backbone-Architektur dargestellt. Einige der dort dargestellten Sensoren wie Kamera und Laser können direkt an das Backbone-Netzwerk angeschlossen werden, da entsprechende Schnittstellen vorhanden sind und die Weiterleitung der Daten an unterschiedliche Bussysteme ermöglicht wird. In Domänen mit speziellen Anforderungen - wie z.B. Antrieb - haben sich Bussysteme etabliert, die nicht ohne weiteres durch neue Technologien ersetzt werden können. Diese können über Gateways an das Backbone-Netzwerk angebunden werden und so mit den restlichen Systemen im Automobil interagieren. Durch die Gateways wird die Grundlage für eine schrittweise Integration des Time-Triggered Ethernet in das Automobil geschaffen. Sofern es die Zeitanforderungen zulassen, können Bussysteme aufgeteilt werden. In Abbildung I interagieren die Bussysteme Fahrdynamik 1 und Fahrdynamik 2 durch Gateways über das Backbone-Netzwerk.

Durch Simulationen können neue Software-Module mit geringen Kosten und auf schnellstem Wege erprobt werden. Die einzelnen Software-Module können variabel ausgetauscht und ergänzt werden. Dadurch ist es möglich Wechselwirkungen zu analysieren und neue Netzwerklösungen zu erproben.

Diese Arbeit erfolgt im Rahmen der Forschungsgruppe (CoRE Research Group)[8]. Es wurde ein Simulationsmodell eines solchen Gateways entwickelt, das für unterschiedliche Bussysteme ausgelegt ist. Dabei wurden mehrere CAN-Bussysteme über ein Gateway an das Time-Triggered Ethernet Backbone-Netzwerk angebunden. Als Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways baut es auf die bestehenden Simulationsmodelle von Time-Triggered Ethernet und CAN auf, die in der Forschungsgruppe CoRE an der HAW Hamburg entwickelt wurden. Die notwendigen Maßnahmen für eine Erweiterung des Gateways auf mehrere, unterschiedliche Bussysteme werden in dieser Arbeit erörtert. Das Simulationsmodell enthält die Möglichkeit zur einfachen Erweiterung um weitere Bussysteme.

[8] CoRE (Communication over Real-Time Ethernet Group)

# 2 Grundlagen

Das Kapitel Grundlagen gibt einen allgemeinen Überblick über Bussysteme und geht ausführlich auf die Themenschwerpunkte CAN und Time-Triggered-Ethernet ein. Ein kurze Einführung in das Bussystem FlexRay ist ebenfalls enthalten. Diese Grundlagen sollen einem besseren Verständnis des Multi-Bus Realtime Ethernet Gateways dienen.

## 2.1 Bussysteme

Bussysteme spielen für die „Automotive“ Anwendungen eine entscheidende Rolle, da sie die Infrastruktur für den notwendigen Datenaustausch bilden. Die Grenzen der „Automotive“ Anwendungen werden häufig durch die verwendeten bzw. vorhandenen Bussysteme bestimmt. Eine fehlertolerante Funktionsweise dieser Systeme ist außerdem sicherheitsrelevant.

### 2.1.1 Topologien

In der Kommunikationstechnologie unterscheidet man zwischen Bussystemen und Netzwerken. In Bussystemen nutzen die Teilnehmer ein gemeinsames Übertragungsmedium, auf dem nur ein Teilnehmer zeitgleich senden kann. Das Übertragungsmedium bildet eine sogenannte Kollisionsdomäne. Sobald ein weiterer Teilnehmer zeitgleich sendet, kommt es zu einer Kollision in der Nachrichtenübertragung. In Netzwerken werden die Kollisionsdomänen teilweise oder vollständig aufgelöst, da alle Teilnehmer durch Punkt-zu-Punkt Verbindungen angeschlossen werden. Diese Punkt-zu-Punkt Verbindungen können eine bidirektionale Übertragung zulassen, sodass die Kollisionsdomänen vollständig aufgelöst werden. Dadurch werden viele parallele Nachrichtenaustausche im Netzwerk möglich.

Sowohl in Bussystemen als auch in Netzwerken können die Verbindungen der Teilnehmer unterschiedlich aufgebaut und strukturiert werden. Nicht alle Topologien passen für alle Bussysteme oder Netzwerke. Häufig unterstützen sie aber mehr als eine Topologie, die für verschiedene Anwendungen Vor- bzw. Nachteile bieten.

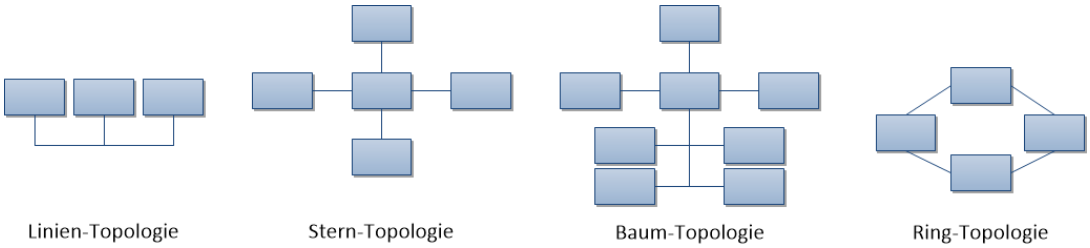


**Abbildung II: Netztopologien (vgl. Wallentowitz, et al., 2006)**

In Abbildung II sind die Struktur und der Aufbau der verschiedenen Topologien dargestellt. Sowohl CAN als auch FlexRay unterstützen Linien- und Stern-Topologie. Switched Ethernet, das aus gewöhnlichen Ethernet-Netzwerken in Büro und Haushalt bekannt ist, basiert auf der Stern-Topologie.

### 2.1.2 Zugriffverfahren

Bei dem Zugriff auf das Übertragungsmedium ist generell zwischen Master-Slave und Multi-Master Architekturen zu unterscheiden. Bei der Master-Slave Architektur gibt es einen Master, der den Zugriff steuert. Bei der Multi-Master Architektur hingegen existieren mehrere Master, die den Zugriff untereinander abstimmen müssen. Der CAN-Bus arbeitet beispielsweise nach der Multi-Master Architektur, wo alle Teilnehmer als Master agieren.

Der Zugriff auf ein Übertragungsmedium kann zeitgesteuert (time-triggered) oder ereignisgesteuert (event-triggered) erfolgen. Bei Multi-Master Architekturen mit ereignisgesteuerter Kommunikation kann es vorkommen, dass mehrere Teilnehmer gleichzeitig Daten übertragen wollen und damit Zugriff auf das Übertragungsmedium wünschen. Dies wird vorwiegend durch das CSMA[9] Zugriffsverfahren geregelt. Zeitgesteuerter Zugriff wird meist durch TDMA[10] organisiert.

**CSMA-Zugriffsverfahren**

Grundsätzlich wird zwischen CSMA/CD (collision detect) und CSMA/CA (collision avoidance) unterschieden. CSMA/CD erkennt eine Datenkollision, kann diese aber nicht verhindern. Bei CSMA/CA werden Datenkollisionen durch sogenannte Arbitrierungsverfahren geregelt, sodass nur ein Teilnehmer die Sendeberechtigung erhält. Die Regelung basiert auf der Definition von einem dominanten und rezessiven Bit-Zustand auf dem Übertragungsmedium. Der hohe Signalpegel, die logische 1, wird als rezessiver Bit-Zustand definiert, während ein niedriger Signalpegel, die logische 0, als dominant definiert wird. Während der Arbitrierungsphase senden alle Teilnehmer, die eine Nachricht senden wollen, ihren Pegel auf das Übertragungsmedium und kontrollieren, ob der Bus-Pegel mit dem selbst gesendeten Pegel übereinstimmt. Sendet ein Teilnehmer eine logische 1 und ein weiterer eine logische 0, so gewinnt stets die dominante logische 0. Der Teilnehmer, der die

[9] CSMA (Carrier Sense Multiple Access)
[10] TDMA (Time Division Multiple Access)

logische 1 sendet, verliert das Arbitrierungsverfahren und versucht später erneut zu senden. Auf diese Weise geht nur ein Teilnehmer als Gewinner aus dem Arbitrierungsverfahren hervor, der daraufhin den Sendevorgang fortsetzt. In Kapitel 2.3.2 wird das Arbitrierungsverfahren am Beispiel des CAN-Bussystems detailliert dargestellt.

**TDMA-Zugriffsverfahren**
Dieses Zugriffsverfahren nutzt statisch gesetzte Zeitfenster, die eine gemeinsam synchronisierte Zeit aller Teilnehmer voraussetzt. Es werden Zeitfenster definiert, in welchen genau ein Teilnehmer senden darf. Stehen dem Teilnehmer keine Daten zur Versendung zur Verfügung, verstreicht sein Zeitfenster ungenutzt. Durch die statisch festgelegten Zeitfenster können genaue Angaben gemacht werden, welche max. Verzögerungszeit eine Nachricht hat. Das Zeitverhalten ist demnach streng deterministisch[11] und besonders für periodische Übertragungen geeignet.

## 2.2 Echtzeitanforderungen

Echtzeitanforderungen in Kommunikationssystemen sind zeitliche Vorgaben an das System, innerhalb dessen eine Reaktion des Systems erfolgen muss. Werden diese Deadlines vom System überschritten, so ist dies eine Fehlfunktion des Systems.

Unterschiedliche physikalische Bedingungen führen zu unterschiedlichen Echtzeitanforderungen. Für Anwendungen in der Motorsteuerung können Zeitvorgaben im µs-Bereich notwendig sein, während Anwendungen in der Komfortsteuerung Zeitvorgaben im unteren ms-Bereich haben.

Zudem wird zwischen harten und weichen Echtzeitanforderungen unterschieden. Kann eine Deadline mit harter Echtzeitanforderung nicht eingehalten werden, so führt dies zu einer Katastrophe, beispielsweise zur Zerstörung des Motors. Wird hingegen eine Deadline mit weicher Echtzeitanforderung nicht eingehalten, so führt dies lediglich zu einer Unannehmlichkeit in der Nutzung der entsprechenden Funktion. Dies könnte zum Beispiel die Regelung der Klimaanlage betreffen.

## 2.3 CAN

Bosch entwickelte Mitte der 80er Jahre in Zusammenarbeit mit Intel das Controller Area Network, kurz CAN, für die Vernetzung von Steuergeräten in Automobilen. Es hat sich in der Automobilindustrie als Standard etabliert und wird vor allem für die Vernetzung von Steuergeräten im Motor- und Getriebemanagement verwendet. In dem Kapitel Physical Layer wird ein grober Überblick über den physikalischen Aufbau gegeben sowie Kennzahlen des CAN-Bussystems dargestellt. Die Funktionsweise des CAN-Protokolls samt Nachrichtenformaten wird im Kapitel Data Link Layer beschrieben.

[11] vorhersagbar

### 2.3.1 Physical Layer

Das CAN-Bussystem unterstützt sowohl Linien- als auch Stern-Topologie. Es hat eine max. Bruttodatenrate von 1 Mbit/s, wobei die Nettodatenrate[12] 58% beträgt. Die max. Länge des Übertragungsmediums ist abhängig von der Datenrate (vgl. Wallentowitz, et al., 2006 S. 205). Bei 1 Mbit/s beträgt die maximale Länge 40m, bei 500 kBit/s maximal 1000m.

Abbildung III zeigt den elementaren Aufbau eines Can-Bussystems mit Zweidrahtleitung in Linien-Topologie. Die Busleitungen CAN_H(CAN-High-Leitung) und CAN_L(CAN-Low-Leitung) sind zur Reduzierung von Störungen durch Reflexionen an den sonst offenen Leitungsenden durch 120 Ohm Widerstände abgeschlossen (vgl. Wallentowitz, et al., 2006 S. 205).

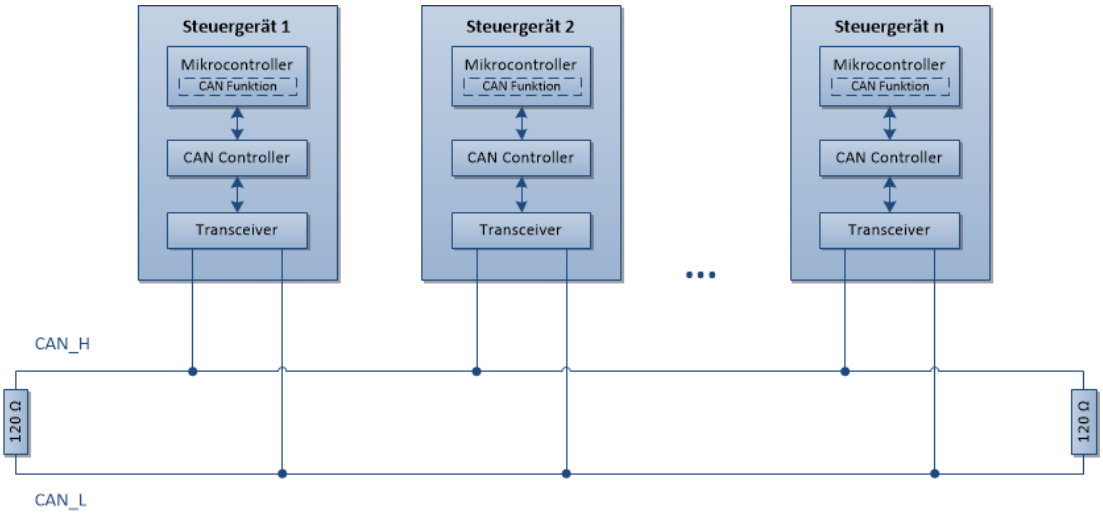


**Abbildung III: CAN-Bus in Linien-Topologie**

Der Transceiver ist für das Senden und Empfangen der elektrischen Signale zuständig. Hierüber werden sowohl binäre Datenströme des CAN-Controllers in elektrische Signale als auch die elektrischen Signale des CAN-Bussystems in binäre Datenströme umgewandelt, die für den CAN-Controller analysierbar sind. Die CAN-Transceiver werden in verschiedenen Kategorien unterschieden. Der Highspeed-CAN (Klasse C) hat eine Bitrate von größer gleich 250 kBit/s und ist nicht fehlertolerant gegen den Ausfall einer Ader der Zweidrahtleitung. Mit einer Bitrate von kleiner gleich 125 kBit/s ist der Lowspeed-CAN(Klasse B) wesentlich langsamer als der Highspeed-CAN, ist aber bei Ausfall einer Ader der Zweidrahtleitung weiterhin funktionsfähig. Der Single-Wire-CAN ist mit der Bitrate von 33 1/3 bzw. 83 1/3 kBit/s der Klasse B zuzuordnen. Der Vorteil liegt hier vor allem in den Kosteneinsparungen durch ungeschirmte Eindrahtleitungen.

[12] Beinhaltet nur Nutzdaten

Der CAN-Controller implementiert die Funktionen des CAN-Protokolls. Es wird zwischen Basic-CAN und Full-CAN Controllern unterschieden. Der Basic-CAN Controller beinhaltet nur Grundfunktionen, während der Full-CAN Controller alle Funktionen des CAN-Protokolls implementiert. Je nach Anwendung kann eine Entlastung des Mikrocontrollers durch einen Full-CAN Controller sinnvoll sein oder aber ein günstiger Basic-CAN Controller, bei dem einige Funktionen des CAN-Protokolls wie Akzeptanz- und Fehlerprüfung (siehe 2.3.2/2.3.3) durch den Mikrocontroller übernommen werden.

### 2.3.2 Data Link Layer

Die ereignisgesteuerte Kommunikation im CAN-Bus erfolgt nach dem Prinzip der Multi-Master Architektur. Alle Steuergeräte sind gleichberechtigt und können ihren CAN-Controller jederzeit mit der Sendung eines Datenframes beauftragen. Trotzdem kann nur ein Steuergerät zurzeit einen Sendevorgang durchführen. Dies wird durch das Arbitrierungsverfahren sichergestellt.

Die Adressierung erfolgt nicht über physikalische Adressen der einzelnen Steuergeräte, sondern jede Nachricht besitzt einen systemweit eindeutigen CAN-Identifier(CAN-ID). Die CAN-ID stellt zusätzlich die Priorität der Nachricht während des Arbitrierungsverfahren dar, wobei die niedrigste Zahl die höchste Priorität hat. Wenn ein Motor-Steuergerät eine Nachricht mit der Motortemperatur schicken soll, so wird dieser Nachricht während der Offline-Konfiguration eine CAN-ID zugewiesen. Somit steht jede CAN-ID für einen bestimmten Nachrichteninhalt. Das Standard Format, CAN 2.0A, hat eine CAN-ID mit 11 Bit, während das Extended Format, CAN 2.0B, eine CAN-ID von 29 Bit hat.

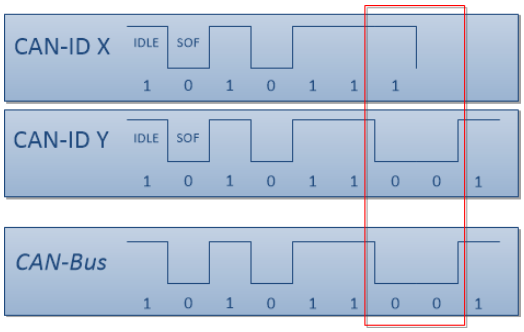


Abbildung IV: Arbitrierungsverfahren

**Arbitrierungsverfahren**

In Abbildung IV ist das Arbitrierungsverfahren mit zwei Teilnehmern dargestellt. Ein Teilnehmer sendet mit der CAN-ID X, ein anderer mit der CAN-ID Y. Auf dem CAN-Bus wird der Pegel des CAN-Bussystems wiedergegeben.

Wie in Abbildung V zu sehen ist, startet ein Frame stets mit dem Start-of-Frame (SOF) Bit, gefolgt von der CAN-ID. Alle Teilnehmer vergleichen während des Sendevorgangs den Bus-Pegel mit dem selbst gesendeten Pegel. Sobald ein Teilnehmer selbst ein rezessives Bit sendet und auf dem Bus ein dominantes Bit erkennt, ist für diesen Teilnehmer die Arbitrierung verloren und er bricht den Sendevorgang ab (siehe rote Markierung in Abbildung IV). Der Teilnehmer mit der höchsten Priorität (geringste CAN-ID) gewinnt die Arbitrierung und sendet den kompletten Frame.

Jedes Steuergerät empfängt alle Nachrichten, die auf den Bus gelegt werden, allerdings sind nicht alle Nachrichten für jedes Steuergerät interessant. Aus diesem Grund besitzt jeder CAN-Controller einen programmierbaren Akzeptanzfilter, dem eine Liste aller relevanten CAN-IDs bereitgestellt wird. Auf diese Weise lässt der CAN-Controller nur Nachrichten zur Steuergerät-Anwendung zu, die von Interesse sind.

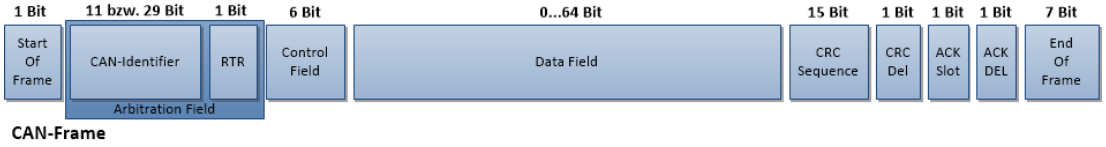


Abbildung V: Aufbau des CAN-Frames

In Abbildung V ist die Struktur eines CAN-Frames dargestellt, die nun vom Aufbau her Schritt für Schritt beschrieben wird.

**Unterscheidung Daten- und Remote Frame**

Prinzipiell haben sowohl Daten- als auch Remote-Frames den Aufbau, wie er in Abbildung V dargestellt ist. Sie unterscheiden sich jedoch in der Nutzung der Felder. Ein Daten-Frame dient zum Transport von Daten und hat ein gefülltes Data Field. Mit einem Remote-Frame kann man einen anderen Teilnehmer auffordern, einen Daten Frame mit der im Remote-Frame genutzten CAN-ID zu senden. Deshalb haben Remote-Frames kein Data Field. In dem Control Field muss allerdings die Länge des Data Fields des gewünschten Data Frames hinterlegt sein. Entscheidend ist das Remote-Transmission-Request(RTR) Field, das bei Daten Frame dominant und bei Remote Frames rezessiv ist.

**Start-of-Frame Field**

Im Leerlaufzustand (IDLE) hat der CAN-Bus einen rezessiven Bus-Pegel. Ein Sendevorgang kann nur im Leerlaufzustand des Bussystems gestartet werden und das Start-of-Frame (SOF) Bit kennzeichnet mit einem dominanten Bit den Anfang des Frames. „Sender und Empfänger synchronisieren sich auf die fallende Flanke des SOF.“ (Wallentowitz, et al., 2006 S. 208)

**Arbitration Field**

Das Arbitration Field besteht aus dem CAN-Identifier und dem Remote-Transmission-Request (RTR)-Bit. Ein rezessives RTR-Bit besagt, dass es sich um einen Remote-Frame handelt, wobei ein dominantes RTR-Bit einen Daten-Frame signalisiert. Versucht ein Steuergerät ein Remote-Frame zu senden, wenn ein anderes Steuergerät zur selben Zeit ein Daten-Frame der gleichen CAN-ID sendet, so wird der Daten-Frame mit dem dominanten RTR Field gesendet, da das RTR Field mit in die Arbitrierung einbezogen wird.

**Control Field**
„Das Control Field besteht aus 6 Bits. Es enthält ein Identifier Extension Bit (IDE), ein reserviertes Bit für zukünftige Erweiterungen und vier Bit für die Anzahl der Daten-Bytes im Data Field … Das IDE-Bit gibt an, ob es sich um einen CAN-Frame im Standard-Format (11 Bit CAN-ID wenn IDE-Bit dominant) oder einen CAN-Frame im erweiterten Format (29 Bit CAN-ID wenn IDE-Bit rezessiv) handelt. Diese Zuordnung führt dazu, dass bei gleichzeitigem Senden einer Nachricht im Standard-Format und einer Nachricht im Extended-Format bei identischem ersten 11 Bit (dem sog. Base Identifier) der Sender mit dem 29-Bit-Identifier die Arbitrierung verliert. Daher ist es zulässig, in einem CAN-Netzwerk sowohl Standard-Frames als auch Extended-Frames zu verwenden.“ (Wallentowitz, et al., 2006 S. 209)

**Data Field**
Das Data Field beinhaltet die Nutzdaten und kann 0-8 Byte lang sein. Die Nutzdaten können auch mehrere Signale wie Motordrehzahl und Motortemperatur enthalten.

**CRC Field**
Das Cyclic Redundancy Check(CRC) Field enthält 15 Bit mit einer CRC-Prüfsumme über den vorausgegangen Frame und einen 1 Bit CRC-Delimiter(Del), der das Feld stets mit einem rezessiven Bit abschließt. Durch die Prüfsumme können Übertragungsfehler erkannt werden.

**ACK Field**
Das ACK Field (Acknowlegde) dient zur Signalisierung einer fehlerfreien Übertragung durch den bzw. die Empfänger. Der Sender sendet im ACK Slot ein rezessives Bit und jeder Empfänger, der den Frame korrekt empfangen hat (unabhängig von dem Akzeptanzfilter), sendet im ACK Slot ein dominantes Bit. Der Sender überwacht während des Sendens ebenfalls den Bus und kann somit feststellen, ob mindestens ein Empfänger den Frame korrekt empfangen hat. Der ACK-Delimiter (Del) ist ein rezessives Bit, das das ACK Field abschließt.

**End-of-Frame (EOF)**
Das Ende eines Frames wird mit dem End-of-Frame (EOF) bestehend aus 7 rezessiven Bits abgeschlossen.

**Interframe-Space**
Nach dem EOF folgt der Interframe-Space, der aus dem Intermission-Field mit drei rezessiven Bits und dem Leerlaufzustand (IDLE) des Buses mit ebenfalls rezessiven Bits besteht. Der Interframe-Space endet, sobald ein Teilnehmer das Start-of-Frame (SOF) Bit sendet.

### 2.3.3 Fehlerbehandlung

Das Auftreten von Störungen oder fehlerbehafteten Komponenten in technischen Systemen sind nicht ungewöhnlich und müssen soweit wie möglich erkannt und eingeschränkt bzw. beseitigt werden. Das CAN-Protokoll nutzt unterschiedliche Fehlererkennungs-Mechanismen, wie Cyclic Redundancy Code (CRC), den Message-Frame-Check, das Acknowledge, das Monitoring und das Bit-Stuffing (vgl. Wallentowitz, et al., 2006 S. 210). Sobald ein CAN-Bus Teilnehmer - in der Regel ein Steuergerät - einen Fehler erkennt, bricht es die laufende Übertragung ab, indem es einen **Error-Frame** sendet.

**Error-Frame**

Ein Error-Frame besteht aus dem Error-Flag mit 6 dominanten Bits sowie dem Error-Delimiter aus 8 rezessiven Bits. Durch das Error-Flag wird die Bit-Stuffing[13]-Regel absichtlich gebrochen, was dazu führt, dass die anderen Steuergeräte ebenfalls einen Fehler erkennen. Der Fehler wird im System propagiert, sodass die Datenkonsistenz systemweit gesichert ist. Auch das Sender-Steuergerät erkennt den Fehler, stellt den Sendevorgang ein und versucht es später erneut.

Das ständige Senden eines Error-Frames würde dazu führen, dass keine Daten mehr ausgetauscht werden können. Um eine Flutung von Error-Frames durch fehlerhafte Steuergeräte zu vermeiden, wechseln die Steuergeräte aufgrund von Fehlerhäufungen in abgestufte Fehlerzustände.

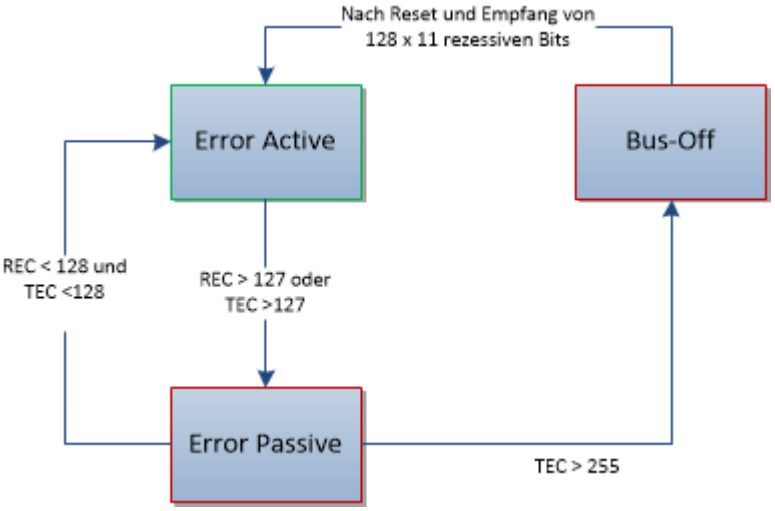


Abbildung VI: CAN Fehlerzustandsdiagramm

In Abbildung VI sind die drei Fehlerzustände des CAN-Protokolls Fehleraktiv (Error Active), Fehlerpassiv (Error Passive) und Bus-Off dargestellt. „Jeder CAN-Knoten besitzt einen Sendefehlerzähler mit der Bezeichnung Transmit Error Counter (TEC) und einen Empfangsfehlerzähler mit der Bezeichnung Receive Error Counter (REC). Die Steuergeräte inkrementieren oder dekrementieren die Zähler nach festen Regeln. Je nach Zählerstand wechselt das Steuergerät seinen Fehlerzustand von fehleraktiv über fehlerpassiv nach Bus-

[13]Siehe Abschnitt Bit-Stuffing auf Seite 19

Off oder zurück.“ (Wallentowitz, et al., 2006 S. 210) Fehleraktive Steuergeräte senden bei Fehlererkennung einen Error-Frame und fehlerpassive 6 rezessive Bits, wodurch sie nicht mehr Nachrichten überschreiben, die durch andere Steuergeräte fehlerfrei empfangen werden. Steuergeräte im Zustand Bus-Off sind je nach Implementierung mehr oder weniger von dem Busverkehr abgeschnitten. Damit dauerhaft defekte Steuergeräte nach einem Bus-Reset den Bus nicht sofort wieder blockieren können, wechselt ein Steuergerät nach dem Bus-Reset erst nach 128 x 11 rezessiven Bits in den fehleraktiven Zustand. Abbildung VI enthält entsprechende Angaben.

**Fehlererkennungs-Mechanismen**
Erkannte Fehler werden durch einen Error-Frame im System propagiert. Im folgendem werden die Anfangs genannten Fehlererkennungs-Mechanismen etwas genauer erläutert.

**CRC-Check**
Der Sender berechnet eine Prüfsumme über bestimmte Teile des Frames und fügt sie dem zu sendenden Frame an. Der Empfänger führt die gleiche Berechnung durch und kann diese mit der übermittelten Prüfsumme des Senders vergleichen. Stimmen diese Werte nicht überein, besteht ein Bit-Übertragungsfehler. Das folgende Zitat beschreibt den Vorgang etwas genauer.
„Daten- und Remote-Frames werden vom Sender mit einer Prüfsumme versehen. Das SOF-Bit, das Arbitration Field, das Control Field und das Data Field werden als Polynom betrachtet und durch ein Generatorpolynom dividiert (Modulo-2-Division). Der Divisionsrest bildet die CRC-Sequenz. Der oder die Empfänger berechnen ihrerseits die CRC-Prüfsumme und vergleichen sie mit der vom Sender übermittelten. Bei fehlerfreier Übertragung ergibt sich dieselbe CRC-Sequenz.“ (Wallentowitz, et al., 2006 S. 211)

**Message-Frame-Check**
Sowohl von Sendern als auch von Empfängern werden die Frame-Struktur und dessen Länge überprüft.

**Acknowlegde**
Der Sender kann durch ein dominantes Bit im ACK Slot feststellen, ob mindestens ein Empfänger den Frame fehlerfrei empfangen hat.

**Monitoring**
Sowohl beim Senden als auch beim Empfangen analysieren die CAN-Teilnehmer den Bus-Pegel. Bitfehler beim Senden können auf diese Weise erkannt werden, wenn Sende- und Bus-Pegel nicht übereinstimmen.

**Bit-Stuffing**
Der CAN-Bus arbeitet mit der Non-Return-to-Zero (NRZ)-Codierung. Um mögliche Synchronisationsprobleme durch seltene Flankenwechsel zu vermeiden, wird nach fünf

gleichwertigen Bits ein komplementäres Bit zusätzlich eingesetzt. Der Empfänger arbeitet nach gleichem Prinzip und führt das „Destuffing“ aus, indem alle zusätzlich eingefügten Bits beim Empfänger wieder entfernt werden. Erkennt ein Empfänger eine Bitfolge von mehr als fünf gleichwertigen Bits, so liegt ein Bit-Stuffing-Fehler vor.

## 2.4 FlexRay

Im Jahr 2000 entwickelten unter anderem BMW und DaimlerChrysler ein neues Bussystem. Es bekam den Namen FlexRay und vereint die Ansätze verschiedener herkömmlicher Bussysteme. Da die existierenden Bussysteme nicht allen Anforderungen der Automobil-Industrie gerecht werden konnten, sollte FlexRay ein Bussystem für sicherheitsrelevante, verteilte Systeme werden. Heute ergänzt es die bereits etablierten Bussysteme im Automobil. Dieses Kapitel gibt einen Überblick über die Funktionsweise von FlexRay.

### 2.4.1 Bus-Topologie und Physical Layer

FlexRay kann sowohl in Linien- als auch in aktiver oder passiver Sterntopologie genutzt werden. Eine Besonderheit von FlexRay ist die Unterstützung von sowohl einkanaligen als auch zweikanaligen Systemen. Bei zweikanaligen Systemen können die Teilnehmer wahlweise an nur einen oder beide Kanäle angeschlossen werden. Die maximale Übertragungsrate beträgt 10 Mbit/s pro Kanal.

### 2.4.2 Data Link Layer

FlexRay nutzt das Zugriffsverfahren TDMA- bzw. FTDMA[14], um den Zugriff auf den Bus zu regeln. Sowohl TDMA als auch FTDMA definieren Zeitschlitze, in denen ein Teilnehmer senden darf. In FlexRay werden Kommunikationszyklen definiert, die ein statisches und ein dynamisches Segment haben. Sowohl im statischen als auch im dynamischen Segment kann wahlweise auf einem oder beiden Kanälen gesendet werden. Das Senden auf beiden Kanälen erhöht die Ausfall- und Übertragungssicherheit. Wird nur ein Kanal verwendet, so kann parallel ein anderer Teilnehmer auf dem anderen Kanal übertragen, sodass eine höhere Bandbreite erreicht wird.

Im statischen Segment erfolgt die Bus-Zuteilung nach TDMA. Die Zeitschlitze haben dort alle dieselbe Länge und sind unveränderbar, wobei insgesamt 64 aufeinanderfolgende Kommunikationszyklen definiert werden können. In dem statischen Segment eines Kommunikationszyklus senden die gleichen Teilnehmer mit derselben Frame-ID im selben Zeitschlitz. Die feste Reihenfolge der Kommunikationszyklen sorgt dafür, dass ein Kommunikationszyklus in festen Zeitanständen durchlaufen wird.

Im dynamischen Segment erfolgt die Bus-Zuteilung nach FTDMA, sodass ein Teil der Bandbreite bedarfsorientiert genutzt werden kann. Im Gegensatz zu TDMA arbeitet FTDMA

[14] FTDMA (Flexible Time Division Multiple Access)

mit einer dynamischen, prioritätsgesteuerten Bus-Zuteilung, die aber auch auf dem Prinzip von Zeitschlitzen beruht. Die Zuteilung des Senderechts erfolgt im dynamischen Segment deshalb prioritätsgesteuert. Nachrichten mit niedriger Priorität kommen möglicherweise nicht sofort zum Zuge. Sowohl das statische als auch das dynamische Segment haben eine feste Länge, sodass auch ein Kommunikationszyklus eine feste Länge aufweist. Nach dem Durchlauf aller Kommunikationszyklen folgt der nächste Durchlauf.


Abbildung VII: Aufbau des FlexRay-Frames

In Abbildung VII ist der Aufbau eines FlexRay-Frames dargestellt. In den 5 Control Bits können Nachrichten indiziert werden, die eine spezielle Bedeutung im Netzwerk haben. Die Frame-ID ist wie die CAN-ID ein eindeutiger Identifier der Nachricht, der gleichzeitig deren Priorität angibt. Auch bei FlexRay referenziert die Frame-ID keinen direkten Teilnehmer, sondern den Inhalt einer Nachricht. Payload-length gibt die Länge des variablen Data-Field an. Header-CRC enthält die CRC-Prüfsumme aller davorliegenden Protokollfelder. Im Cycle-Count wird der aktuelle Wert des Kommunikationszyklus angegeben, der mit jedem durchlaufenden Kommunikationszyklus inkrementiert wird. Im Data-Field werden die Nutzdaten mitgegeben, die 0-254 Byte lang sein können. Die Einteilung erfolgt allerdings in Daten-Worten bestehend aus 16-Bit, sodass 0-127 Datenworte enthalten sein können. Das CRC-Feld enthält die CRC-Prüfsumme um den Frame vor Übertragungsfehlern zu schützen.

## 2.5 Time-Triggered Ethernet

Dieses Kapitel beinhaltet eine ausführliche Einführung in das Time-Triggered Ethernetprotokoll und soll zur Orientierung und zum Verständnis der Arbeit dienen. Detaillierte Informationen sind in der Time-Triggered Ethernet Spezifikation zu finden (vgl. SAE International Group, 2011).

Time-Triggered Ethernet basiert auf einem Forschungsprotokoll der Technischen Universität Wien und wurde später durch die (TTTech Computertechnik AG) weiterentwickelt. Es wurde unter anderem unter Berücksichtigung der Anforderungen im Automotive Kontext erarbeitet.

Time-Triggered Ethernet (TTEthernet) baut auf das Switched Ethernet nach IEEE 802.3 auf. Auf der Grundlage des Switched Ethernet definiert TTEthernet synchrone Time-Triggered Services zur zeitgesteuerten Übertragung von Nachrichten unter Einhaltung strikter Zeitvorgaben. Hinzu kommt die Möglichkeit zur Einbindung von asynchronen Services wie

Bandbreitenreservierung und Standard Ethernet mit Best-Effort Technologie. Alle Services sind in die drei Nachrichtenklassen Time-Triggered(TT), Rate-Constrained(RC) und Best-Effort(BE) aufgeteilt und werden in Kapitel 2.5.2 detailliert erklärt.

TTEthernet bietet die Möglichkeit, sämtliche Zeitanforderungen - sowohl time-triggered als auch event-triggered - über denselben physikalischen Link abzuwickeln. Applikationen in einem Automobil z.B. Motorsteuerung- und Infotainment-Applikation können somit dasselbe physikalische Netzwerk verwenden. Das Spektrum reicht von strikt deterministischen Zeitvorgaben bis zu Standard Ethernet, bei welchem weder Zeitgarantien noch Übertragungsgarantien gegeben sind.

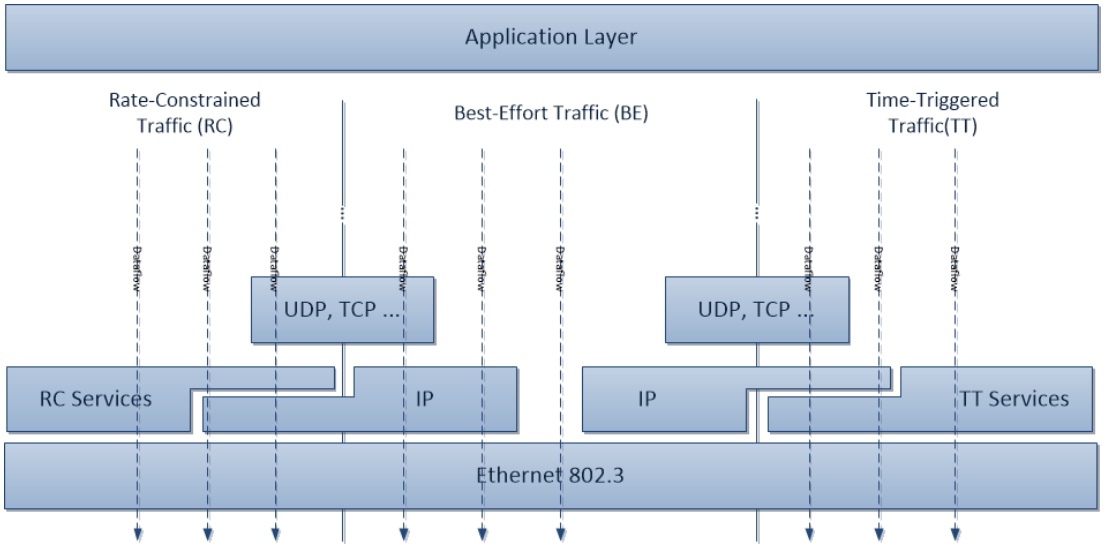


**Abbildung VIII: Zusammenspiel von Standards und Ethernet Protokoll Schichten. (vgl. SAE International Group, 2011)**

TTEthernet vereint unterschiedliche Standards in einem Netzwerk. Der Best-Effort Traffic stammt aus dem Switched Ethernet. Rate-Constrained Traffic dagegen stammt aus AFDX[15] und Time-Triggered Traffic ist neu spezifiziert worden. In Abbildung VIII ist das Zusammenspiel von Standards und den Protokollschichten dargestellt, wie sie in TTEthernet vorzufinden sind. Die Applikation kann unterschiedliche Quality-of-Services(QoS) für die Kommunikation verwenden. Die Pfeile zeigen den Datenfluss durch die unterschiedlichen Schichten. Eine IP oder UDP Nachricht kann beispielsweise als time-triggered Message weitergeleitet werden. Um RC- bzw. TT-Services zu verwenden, muss entsprechende Hardware bzw. Software eingesetzt werden. Nicht zuletzt durch Cluster-Architekturen können auch TTEthernet und Standard Ethernet Komponenten mit Einschränkungen in einem Netzwerk verwendet werden.

[15] AFDX (Avionics Full Duplex Switched Ethernet (ARINC-Standard 664))

Zusätzlich bringt TTEthernet viele Eigenschaften mit, die aus Switched Ethernet bekannt sind. Beispiele sind hohe Skalierbarkeit sowie hohe Bandbreiten von bis zu 100 Gbit/s. Ein weiterer Vorteil ist die Auflösung der Kollisionsdomänen durch Full-Duplex Punkt-zu-Punkt Verbindungen zwischen den Netzwerk-Teilnehmern, sodass auch parallele Übertragungen möglich sind.

Durch fortlaufende Entwicklung und Verbreitung von TTEthernet-Komponenten können niedrige Kosten in der Anschaffung erreicht werden, nicht zuletzt weil sämtliche Bereiche im Automobil mit dieser Technologie abgedeckt werden können. Hinzu kommt, dass die Kompetenz der Entwickler in Richtung Ethernet weitgehend etabliert ist. Dadurch, dass nur ein Netzwerk für sämtliche Anwendungen benötigt wird, sinkt zudem die Komplexität der Systeme und die Interaktion der interdisziplinären Anwendungen ist ohne weiteres möglich.

### 2.5.1 Begriffserklärung

**Latenzzeit**

"Die Zeit zwischen der Initialisierung einer Datenübertragung durch einen Sender (Sendewunsch) und dem Eintreffen der Nachricht beim Empfänger ist die Latenzzeit. Systeme mit vorhersagbarer Latenzzeit werden als deterministisch bezeichnet." (Wallentowitz, et al., 2006 S. 199) Die Latenzzeit ist demnach eine Zeitangabe, die besagt, wie lang eine Datenübertragung dauert. Während der Übertragung kann es zu verschiedenen Verzögerungen z.B. im Switch oder Gateway kommen, die die Latenzzeit beeinflussen. Sind diese Verzögerungen vorhersagbar, so ist auch die Latenzzeit vorhersagbar.

**Jitter**

Der Jitter gibt die Schwankungen der Latenzzeit an. Die Differenz zwischen der worst-case und best-case Latenzzeit wird als Jitter bezeichnet.

### 2.5.2 Nachrichtenklassen

Wie in Abbildung VIII zu sehen ist, unterscheidet TTEthernet zwischen den drei Nachrichtenklassen Time-Triggered (TT), Rate-Constrained (RC) und Best-Effort (BE). Nachrichten der Klasse TT besitzen die höchste Priorität, RC die mittlere und Best-Effort die geringste Priorität. Unabhängig vom Zeitpunkt des Eintreffens der Nachricht wird immer die Nachricht der Nachrichtenklasse mit höchster Priorität zuerst versendet. Alle Nachrichten werden in Bezug auf ihre Priorität versendet.

**Time-Triggered (TT)**

Time-Triggered (TT) Nachrichten werden zu festen Zeitpunkten versendet, die sich zyklisch wiederholen. Das Scheduling aller Time-Triggered Nachrichten wird offline während des Entwicklungsprozesses definiert. Time-Triggered Nachrichten erfüllen strikte

Echtzeitanforderungen und sind für Anwendungen wie X-by-wire[16] vorgesehen. Durch die globale Uhrensynchronisation können die Nachrichten mit einer Genauigkeit im unteren Mikro-Sekunden Bereich übertragen werden. Mit welcher Genauigkeit das System arbeitet, wird allerdings durch die Genauigkeit der Uhren der einzelnen Komponenten bestimmt.

Der Sender einer TT-Nachricht definiert einen Sendezeitpunkt. Der Empfänger definiert einen Empfangszeitraum, indem die Nachricht empfangen werden darf sowie einen Zeitpunkt ab wann die Nachricht zur weiteren Verarbeitung zur Verfügung steht. Nachrichten, die außerhalb des Empfangszeitraumes eintreffen, werden als ungültig bzw. fehlerhaft wahrgenommen. Dadurch entsteht ein wirksamer Sicherheitsmechanismus, der fehlerhafte Übertragungen von TT-Nachrichten aufdeckt.

Der Designer des TTEthernet Netzwerks muss das Scheduling der TT-Nachrichten so planen, dass es konfliktfrei ablaufen kann. Wird eine geplante TT-Nachricht nicht versendet, da beispielsweise keine neuen Sensordaten vorliegen, gibt TTEthernet die Bandbreite für die anderen Nachrichtenklassen frei.

**Rate-Constrained (RC)**
Rate-Constrained Nachrichten werden nicht zeitgesteuert sondern event-gesteuert versendet. Die Limitierung der Bandbreite erfolgt durch die Festlegung eines minimalen Zeitabstandes zwischen dem Versand zweier Nachrichten. Durch die event-basierte Versendung dieser Nachrichten können mehrere Applikationen zeitgleich über dasselbe Medium einen Sendeversuch starten. Diese Nachrichten werden in Queues vorgehalten, bis sie versendet werden können. Dadurch kann es zu Verzögerungen kommen. Die maximale Latenzzeit kann allerdings statisch berechnet werden, sodass sichere Zeitangaben erreicht werden können.

**Best-Effort (BE)**
Diese Nachrichtenklasse beinhaltet Standard Ethernet Nachrichten. Für diese Nachrichten gibt es keine Garantie ob und wann sie gesendet werden. Die Bandbreite, die weder für TT noch für RC Nachrichten verwendet wird, kann für BE Nachrichten verwendet werden. Es muss allerdings in der Netzwerkplanung genug Bandbreite für BE Nachrichten vorgesehen werden, wenn Applikationen diese Nachrichtenklasse sinnvoll nutzen sollen.

Über BE Nachrichten können TTEthernet und Standard Ethernet Komponenten miteinander kommunizieren.

[16] Ersetzen der mechanischen Systeme zur manuellen Steuerung durch elektronische Systeme und Bedienelemente.

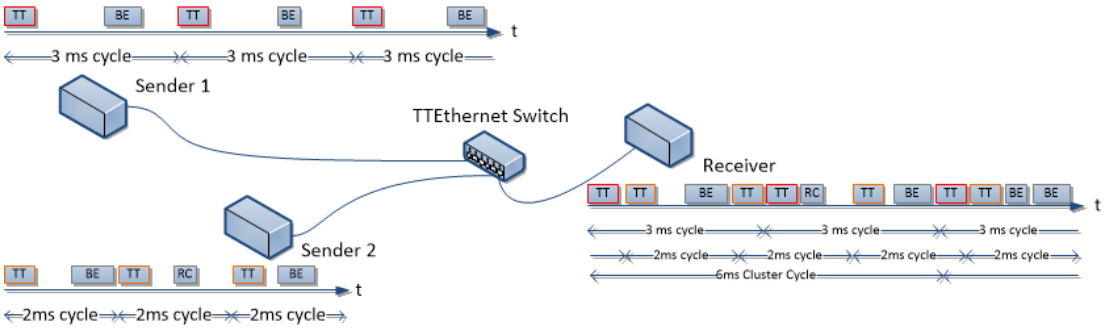


**Abbildung IX: Datenfluss Integration durch einen TTEthernet Switch (vgl. TTTech Computertechnik AG, 2008)**

Abbildung IX zeigt den Datenfluss von zwei Sendern über einen TTEthernet Switch zu einem Receiver. Sender, Receiver und Switch müssen eine Erweiterung der Standard Ethernet Komponenten in Hard- oder Software implementieren, um per TTEthernet mittels RC- und TT-Nachrichten zu kommunizieren. Sender 1 sendet TT-Nachrichten mit einem Zyklus von 3ms und event-basierte BE-Nachrichten. Sender 2 sendet ebenfalls TT-Nachrichten, jedoch in einem 2ms Zyklus. Zusätzlich sendet dieser Sender RC- und BE-Nachrichten. In dem Beispiel wurde ein gemeinsamer Zyklus von 6ms festgelegt und die Sende-Zyklen durch initiale Offsets aufeinander abgestimmt. So wird eine Kollision der zeitgesteuerten Nachrichten im Switch vermieden. Sobald Bandbreite zur Verfügung steht, werden RC- und BE-Nachrichten mit eingeschleust, wobei der Link für anstehende TT-Nachrichten rechtzeitig freigehalten wird. In dem Beispiel können nicht alle BE Nachrichten gesendet werden.

### 2.5.3 Zeitsynchronisation

In dem TTEthernet Netzwerk ist die Zeitsynchronisation von großer Bedeutung, da die TT-Nachrichten im Rahmen der globalen Zeit versendet werden. Die präzise Uhrensynchronisation der verteilten Endsysteme und Switche ist hoch komplex und bildet den Hauptteil der TTEthernet Spezifikation. Dieses Kapitel gibt einen groben Überblick über die Funktionsweise der Zeitsynchronisation. Detaillierte Informationen sind in der Time-Triggered Ethernet Spezifikation (SAE International Group, 2011) zu finden.

Für die Uhrensynchronisation erhält jeder Netzwerk-Teilnehmer eine der drei Rollen Synchronization Master, Compression Master oder Synchronization Client.

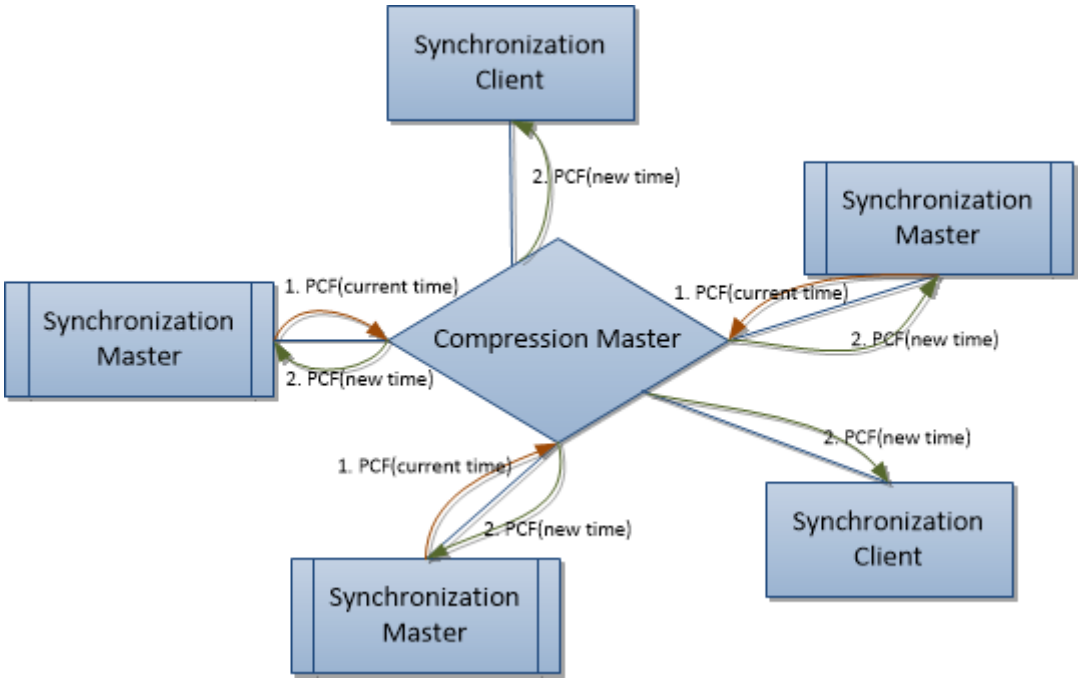


Abbildung X: Zweistufiger Synchronisationsprozess von TTEthernet

Über sogenannte Protocol Control Frames (PCF) wird die Information zur Synchronisation zwischen den Teilnehmern ausgetauscht. In Abbildung X ist der zweistufige Synchronisationsprozess mit dem entsprechenden Datenfluss der PCFs dargestellt. Die Synchronization Master senden zu fest definierten Zeiten einen PCF an den Compression Master. Das stellt die erste Stufe des Synchronisationsprozesses dar. Durch die fest definierten Synchronisationszyklen können Teilnehmer auch in ein laufendes System einsteigen und synchronisiert werden. Nach Ankunft der PCFs, berechnet der Compression Master den Mittelwert aller eingegangenen Zeiten und sendet die neue gemeinsame Zeit an alle Teilnehmer. Dies ist die zweite Stufe des Synchronisationsprozesses. TTEthernet unterstützt auch den Einsatz von mehreren Compression Master, da dieser sonst einen single point of failure[17] bilden würde.

### 2.5.4 Datenframe-Format und Adressierung

Die Übertragung von Daten im TTEthernet Netzwerk erfolgt durch Ethernet-Frames. BE Nachrichten nutzen das Standard Ethernet Format. Um TT- und RC-Nachrichten zu erkennen, wird das *Destination Address* Feld anders interpretiert.

[17] Ausfall einer Komponente führt zum Versagen des gesamten Systems

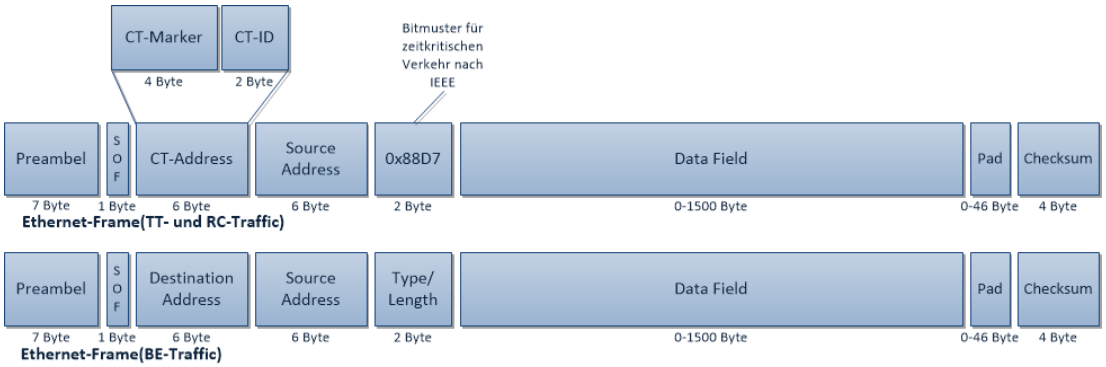


**Abbildung XI: Datenframeformat in TTEthernet**

Das Standard Ethernet verwendet in der *Destination Address* Mac-Adressen zur eindeutigen Identifizierung von Teilnehmern und unterstützt UniCast, MultiCast und BroadCast. Zur eindeutigen Identifizierung von TT- und RC-Nachrichten mit entsprechend höherer Priorität definiert TTEthernet ein eigenes Adressformat, eine Critical Traffic Adresse (*CT-Adress*). Der Unterschied besteht nicht in der Kodierung, sondern in der Interpretation des Adress-Feldes. Wie in Abbildung XI dargestellt, wird die *CT-Address* in den *Critical Traffic Marker*(CT-Marker) und *Critical Traffic ID*(CT-ID) unterteilt. Zusätzlich wird der zeitkritische Verkehr nach IEEE 802.3 durch ein Bitmuster in dem Typ Feld markiert. Durch den CT-Marker kann eine TT- bzw. RC-Nachricht mittels Bit-Masken effizient erkannt werden. Einem CT-Marker, der beispielsweise den Adressraum für die Motorsteuerung definiert, gehört eine Vielzahl an CT-IDs an, die eine Nachricht eindeutig identifizieren. Die Identifizierung erfolgt inhaltsorientiert, d.h. eine CT-ID steht für einen bestimmten Inhalt in der Nachricht z.B. der Motortemperatur. Alle TTEthernet-Teilnehmer definieren, welche CT-IDs empfangen und welche ignoriert werden sollen. Die *CT-Address* verhält sich ähnlich wie MultiCast-Adressen in Standard Ethernet.

### 2.5.5 Virtuelle Links

Die Netzwerkkomponenten wie Steuergeräte oder Switche eines TTEthernet Netzwerkes sind wie im Standard Ethernet mit einem physikalischem Link verbunden. Über diese physikalischen Links verlaufen eine Vielzahl an virtuellen Links, die zu unterschiedlichen Nachrichtenklassen gehören können.

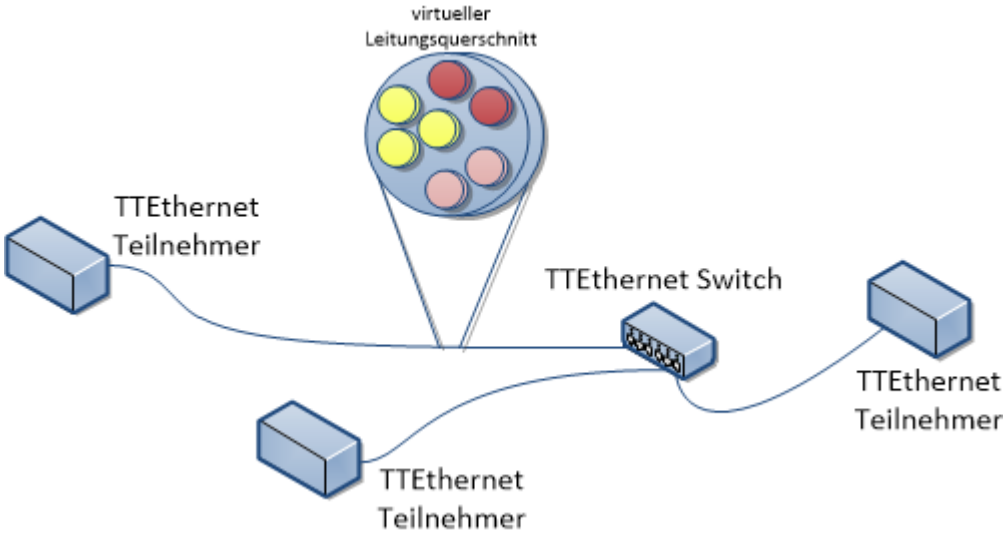


Abbildung XII: Virtueller Leitungsquerschnitt eines physikalischen Links

In Abbildung XII ist der virtuelle Leitungsquerschnitt eines physikalischen Links dargestellt. Die roten virtuellen Leitungen symbolisieren zwei virtuelle Links der Nachrichtenklasse TT, die orangefarbenen Leitungen zwei virtuelle Links der Nachrichtenklasse RC und die gelben Leitungen drei virtuelle Links der Nachrichtenklasse BE. Mit Hilfe der virtuellen Links werden virtuelle Pfade definiert, die von einem Sender zu einem oder mehreren Empfängern reichen. Diese Pfade sind gerichtet und auf einen Nachrichtentyp begrenzt, es kann eine TT Nachricht nur auf einem virtuellen TT Link versendet werden. Der Grund dafür ist, dass die zeitlichen Anforderungen an die virtuellen Links gebunden werden. Ist eine bi-direktionale Verbindung vorgesehen, so muss für jede Strecke ein virtueller Link definiert werden.

Dieses Modell ist lediglich zum Verständnis des Netzwerkaufbaus gedacht und soll als Hilfe zum Verständnis der Netzwerkkonfiguration dienen. Die technische Umsetzung der virtuellen Links erfolgt allein durch Netzwerk-Adressen. Für virtuelle Links der Nachrichtenklasse BE stehen UniCast, MultiCast und BroadCast mit Hilfe von Mac-Adressen zur Verfügung. Für die virtuellen Links der Nachrichtenklassen TT und RC steht nur MultiCast zur Verfügung, da ein speziell interpretiertes Adressformat, die CT-Adressen verwendet wird.

## 2.6 Aufgabe eines Gateways

Die Aufgabe eines Gateways ist es, unterschiedliche Kommunikations- bzw. Informationssysteme miteinander zu verbinden. Grundlegend müssen Protokoll Transformationen von der einen Nachrichten-Darstellung in die andere stattfinden.

Gateways werden häufig in den oberen Schichten des OSI-Modells[18] eingeordnet, da sie in sich geschlossene Systeme miteinander verbinden. Durch das Gateway werden die Grenzen eines Kommunikations- bzw. Informationssystems erweitert, sodass eine globale Kommunikation eines großen inhomogenen Systems möglich wird.

[18] OSI-Modell (Open Systems Interconnection Model)

# 3 Anforderungen an das Gateway

In diesem Kapitel werden Anforderungen an das Multi-Bus Realtime Ethernet Gateway definiert. Hierbei geht es um funktionale Anforderungen an die generellen Funktionen des Gateways, sowie spezifische Anforderungen an Bussysteme. Anforderungen zum Aufbau und zur Verwendbarkeit des Gateways sind in den nichtfunktionalen Anforderungen dargestellt.

Um einen groben Überblick über das Gateway zu verschaffen, wird die Funktion des Multi-Bus Realtime Ethernet Gateways (im Folgenden Gateway genannt) anhand eines kleinen Beispiels in Abbildung XIII dargestellt. Das TTEthernet Backbone-Netzwerk besteht aus zwei Gateways, einem Switch und einem Steuergerät, das direkt an das Backbone-Netzwerk gekoppelt ist. Interessant sind die Gateways, die jeweils mehrere unterschiedliche Bussysteme einbinden.

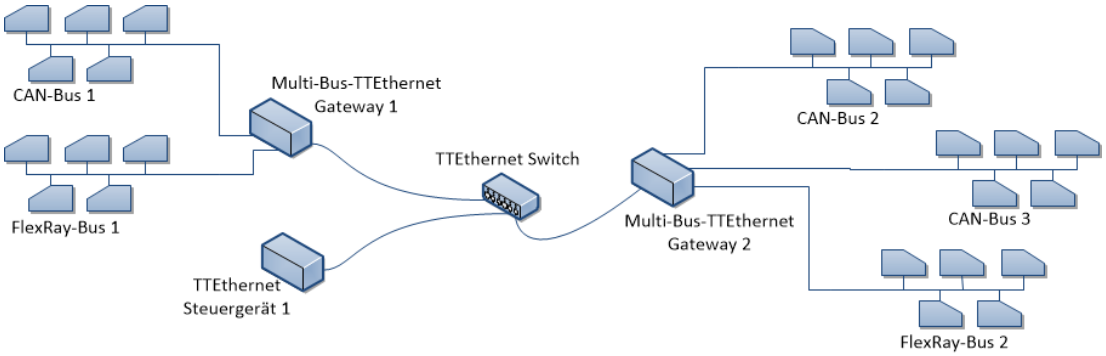


**Abbildung XIII: Netzwerk mit TTEthernet Backbone**

Durch die Gateways ist es nun möglich, eine Nachricht von CAN-Bus 1 an CAN-Bus 2 zu senden. Darüber hinaus können Nachrichten von CAN-Bus 1 an FlexRay-Bus 2 geschickt werden, da das Gateway über geeignete Transformationsroutinen verfügt. Das Senden von Nachrichten zwischen Bussystemen, die am selben Gateway angeschlossen sind, ist ebenfalls möglich.

## 3.1 Funktionale Anforderungen

Aus der beispielhaft dargestellten Funktionsbeschreibung eines Multi-Bus Realtime Ethernet Gateways (Abbildung XIII), ergibt sich eine Menge funktionaler Anforderungen, die in diesem Abschnitt genauer beschrieben werden.

**Routing der Nachrichten**
Das Gateway muss sicherstellen, dass Nachrichten von einem Bussystem zuverlässig an das Ziel-Bussystem übertragen werden. Dazu ist eine statische Routing-Tabelle nötig, in der alle Parameter gespeichert werden können, die für die Übermittlung einer Nachricht eines Bussystems über das TTEthernet Backbone-Netzwerk in ein anderes Bussystem notwendig sind. Insbesondere soll die Konfiguration der Übertragungsart, also TT, RC oder BE, spezifiziert werden können.

**Transformation der Nachrichten**
Das Gateway muss über Transformationsmechanismen verfügen, die es erlauben, Nachrichten in andere Formate zu übertragen. Dazu soll eine Transformation in ein einheitliches Transportformat zur Übertragung über das TTEthernet Backbone-Netzwerk verwendet werden. Für jeden Bus-Frame muss folglich eine Transformation von der Quelldarstellung in die Transportdarstellung und von der Transportdarstellung in die Zieldarstellung transformiert werden.

**Nachrichten-Akkumulation**
Nachrichten des TTEthernet Backbone-Netzwerkes haben eine wesentlich höhere Nutzdaten-Kapazität als Bus-Systeme wie CAN und FlexRay. Gleichzeitig haben die Nutzdaten eine minimale Länge von 46 Byte. Da die Übertragungsrate höher ist als bei gängigen Bussystemen, bietet es sich an, mehrere Bus-Frames in einer TTEthernet Backbone-Nachricht zu verschicken, um die Netzauslastung möglichst gering zu halten.

**Zeitanforderung**
Es muss eine Aussage über die zusätzliche Verzögerungszeit erfolgen, insbesondere in Bezug auf die Pufferung im Gateway. Dies ist notwendig, damit der Netzwerk-Designer das Netzwerk planen kann.

**Synchronisation zwischen Bussystem und TTEthernet Backbone-Netzwerk**
Die Bussysteme und das TTEthernet Backbone-Netzwerk haben keine gemeinsame Uhr. Beide Systeme sollen auch weiterhin unabhängig voneinander arbeiten. Für jedes Bussystem muss ein Konzept zur Einschleusung der Bus-Frames in den Ziel-Bus erstellt werden.

**Fehlerbehandlung**
Fehler, die während der Simulation im Gateway auftreten, müssen durch aussagekräftige Fehlermeldungen dargestellt werden.

**Anforderungen zur Nutzung von CAN-Bussystemen**
Es sollen sowohl Datenframes als auch Remoteframes durch das Gateway übertragen werden. Mindestens CAN-ID, RTR-Flag und Daten-Feld eines CAN-Frames sollen durch das Gateway übertragen werden.

Bussysteme sind in sich geschlossen und das Gateway partizipiert als ein Busteilnehmer. Errorframes sollen ignoriert werden, da sie nur für ein lokales Bussystem Bedeutzung haben und nur dort auswertbar sind. Eine Weiterleitung der Fehler durch das Gateway an andere in sich geschlossene Bussysteme ist deshalb nicht vorgesehen. Diese Fehler können durch die Applikationen der Busteilnehmer abgefangen werden, da dort das Fehlverhalten durch Abgleich des Soll-und Ist-Datenverkehrs erkannt werden kann. Das Verhindern von Folgefehlern wird dadurch sichergestellt.

## 3.2 Nichtfunktionale Anforderungen

In dieser Arbeit soll ein Konzept entwickelt werden, das gleichzeitig mehrere, unterschiedliche Bussysteme und damit verschiedene Nachrichtendarstellungen an einem Gateway zulässt. Im Simulationsmodell wird zunächst ein CAN-TTEthernet Gateway erstellt. Das Simulationsmodell soll leicht um weitere Bussysteme erweitert werden können. Die Anpassungsfähigkeit und Erweiterbarkeit des Gateways soll durch eine geschachtelte Modulstruktur sichergestellt werden.

# 4 Konzept des Gateways

In diesem Kapitel wird die Entwicklung des Konzeptes eines Multi-Bus Realtime Ethernet Gateways beschrieben. Eine grobe Struktur ergibt sich aus der Architektur des Gateways: Die Anbindung der Bussysteme, Routing, Transformation und Pufferung. In diesem Konzept werden vorwiegend die methodischen Aspekte in Betracht gezogen und weniger die physikalische Darstellung der Bitströme, da dies für das Simulationsmodell weitgehend unbedeutend ist. Generell wurde versucht, die Komplexität des Gateways möglichst gering zu halten, um damit eine einfache Anwendung zu gewährleisten.

Einleitend werden einige Begriffe im Zusammenhang mit der Transformation definiert:

**Protokolltransformationen**
Protokolltransformation bedeutet hier die Konvertierung der Nachrichtendarstellung eines Nachrichtenprotokolls in eine Nachrichtendarstellung eines anderen Nachrichtenprotokolls. Beispielsweise ist die Konvertierung eines CAN-Frames in ein Ethernet-Frame eine Protokolltransformation.

**Protokollfeld**
Ein Protokollfeld beschreibt einen spezifischen Teil der Nachrichtendarstellung. Ein CAN-Frame zum Beispiel besteht unter anderem aus den Protokollfeldern CAN-ID und Datenfeld.

**Protokollfeldsequenz**
Eine Protokollfeldsequenz ist eine Folge von mehreren Protokollfeldern, wobei die Reihenfolge keine Rolle spielt, solange eine eindeutige Identifizierung der einzelnen Protokollfelder möglich ist.

## 4.1 Abgrenzung zu vorangegangenen Arbeiten

Die Projektarbeit von Jan Depke [19]befasst sich mit Protokolltransformationen und bedient sich dabei formaler Beschreibungsmethoden. Aus diesem theoretischen Ansatz wurde die Idee von Protokollschichten und Protokollfeldsequenzen zur Transformation von Nachrichten in unterschiedliche Nachrichtendarstellungen übernommen. Nicht zuletzt aufgrund der objektorientierten Programmierung des Simulationsmodelles konnte ein weniger striktes Konzept zur Transformation entwickelt werden. Zum einen konnte die

[19] *Spezifikation von Protokolltransformationen für automotive Anwendungen.* (Depke, 2013)

Komplexität verringert werden, zum anderen wird auch die praktische Umsetzung des Simulationsmodells erleichtert. Die notwendigen Daten zur Transformation wurden in die Routing-Tabelle integriert. Die Begriffsdefinition, die Jan Depke verwendet, weicht in einigen Teilen von den oben beschriebenen Begriffen ab.

## 4.2 Komponenten-Architektur des Gateways

Einleitend wird eine Übersicht über die Komponenten-Architektur gegeben, um den Aufbau und die Struktur des Gateways darzustellen. Die komponentenübergreifende Kommunikation innerhalb des Gateways sowie die Verteilung der Aufgaben werden hier diskutiert.

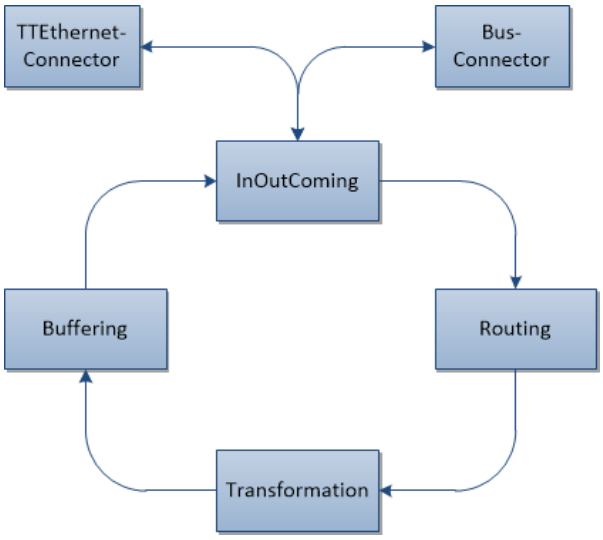


Abbildung XIV: Architektur des Gateways

In Abbildung XIV ist schemenhaft die Architektur des Gateways dargestellt. Der TTEthernet-Connector stellt die Verbindung zum TTEthernet Backbone-Netzwerk dar, während der Bus-Connector sämtliche Bus-Anbindungen verwaltet. Die Komponente InOutComing verbindet die Konnektoren mit den Gateway-Komponenten Routing, Transformation und Buffering. Die Komponenten Routing, Transformation und Buffering werden in den folgenden Kapiteln detailliert erklärt. InOutComing bereitet sowohl eingehende Nachrichten der Konnektoren für die weitere Verarbeitung in den Gateway-Komponenten als auch ausgehende Nachrichten der Gateway-Komponenten für die Connectoren vor. Auf diese Weise wird eine klare Aufgabenteilung durch die Komponenten-Architektur geschaffen, sodass einzelne Komponenten leicht austauschbar und anpassbar sind.

Wie aus der Abbildung XIV deutlich wird, läuft jede Nachricht durch die Gateway-Komponenten Routing, Transformation und Buffering. Dies ist eine Konsequenz der strikten Aufgabenteilung, sodass jede Komponente für sich entscheidet, wie die Nachricht bearbeitet wird. Die Reihenfolge ergibt sich aus dem notwendigen Datenfluss. Die Routing-Informationen werden für die Transformation benötigt und eine Pufferung von Nachrichten ist erst dann sinnvoll, wenn sie vollständig bearbeitet und für den Versand bereit sind.

Die Interaktion zwischen den Komponenten kann durch einen gemeinsamen Speicher oder durch Nachrichtenaustausch gehandhabt werden. Ein Vorteil des Nachrichtenaustausches ist, dass er gleichzeitig eine Triggerung der nachfolgenden Komponente erlaubt. Hier sei die

sogenannte Simulationsumgebung OMNeT++[20] erwähnt. Sie basiert auf Nachrichtenaustausch (engl. Message passing) und bietet hervorragende Möglichkeiten zur Nachrichtendefinition, -versendung und –verarbeitung (siehe Kap. 5). Deshalb wird im Gateway die Interaktion durch Nachrichtenaustausch umgesetzt. Es wird ein eigener Nachrichtentyp für die Kommunikation definiert, sodass eine Nachricht von Komponente zu Komponente weitergereicht werden kann und so die notwendige Information entnommen sowie weitere Informationen hinzugefügt bzw. ausgetauscht werden können.

## 4.3 Routing

Die Intention des Routings ist es, ein Bus-Frame von einem Source-Gateway über das TTEthernet Backbone-Netzwerk zu einem oder mehreren Destination-Gateways zu leiten. (siehe Abbildung XIII: Netzwerk mit TTEthernet Backbone)

Das Routing wird in der Regel mit statischen oder dynamischen Routing-Tabellen durchgeführt. Die gesamte Kommunikation eines Automobil-Netzwerkes wird bereits in der Design-Phase spezifiziert. Deshalb wird mit statischen Routing-Tabellen gearbeitet, die eine präzise Planung des Netzwerkes ermöglichen.

Mit FIBEX[21] kann die gesamte Kommunikation sämtlicher Bussysteme eines Autonetzwerkes durch ein einheitliches XML-Beschreibungsformat definiert werden. FIBEX ist sehr komplex und wird für die einfache Beschreibung des Gateways nicht benötigt. Autohersteller verwenden häufig eine Kommunikationsmatrix (siehe Kap. 4.3.1) in einem Excel-Format. Für die Konfiguration des Gateways wird in dieser Arbeit eine eigene XML-Konfigurationsvorschrift entwickelt, die sich einfach und intuitiv konfigurieren lässt.

### 4.3.1 Aufbau und Funktion der Routing-Tabelle

In der Automobilindustrie wird die Kommunikation zwischen den Netzwerk-Teilnehmern bereits in der Designphase in einer sogenannten Kommunikationsmatrix (K-Matrix) festgelegt. Es existiert keine einheitliche Definition der K-Matrix zwischen den Automobilherstellern, aber grundsätzlich sind Abbildungen von Quell- auf Zielinformationen darin enthalten. Dies bildet u.a. einen Teil unserer Routing-Tabelle. Ein weiterer Teil besteht aus Informationen zum Datenfluss innerhalb des TTEthernet Backbone-Netzwerkes und der Konfiguration von Zusatz-Optionen.

Die Routing-Tabelle ist in die drei übergeordneten Bereiche „source“, „destination“ und „gatewayOptions“ gegliedert. Eine vollständige Übersicht der Parameter ist in Abbildung XV

[20] OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators (OMNeT++ Community)

[21] FIBEX (Field Bus Data Exchange Format (Association for Standardisation of Automation and Measuring Systems, 2013))

dargestellt. Die Dokumenttypdefinition (DTD) ist im Anhang A zu finden. Die Angaben in den eckigen Klammern beschreiben die Kardinalitäten im Format regulärer Ausdrücke, die pro Eintrag in der Routing-Tabelle gelten. Die Bereiche „source“ und „destination“ stellen die Abbildungen von Quell- auf Zielinformationen dar. Die BusID entspricht einem eindeutigen Identifier zur Identifizierung des Bussystems und die ObjectID ist ein Identifier einer bestimmten Nachricht. Durch das Tupel aus BusID und ObjectID kann eine Nachricht im ganzen System eindeutig identifiziert werden. Das ist für die Versendung von Bus-Frames über Busgrenzen hinweg notwendig. Anhand des Type-Feldes wird der entsprechende Transformationsvorgang ermittelt.

| **source[1]** | **destination[+]** | **gatewayOptions[1]** |
|---|---|---|
| sourceBusID | destinationBusID | allwaysSendOldAndNew |
| sourceObjectID | destinationObjectID | |
| sourceType | destinationType | |
| | **Backbone[*]** | |
| | backboneCTID/directMacAdress | |
| | backboneTransferType | |
| | messageAccumulation | |
| | holdUpTime | |

**Abbildung XV: Parameter der Routing-Tabelle**

Um eine Nachricht über Busgrenzen hinweg weiterleiten zu können, sind zusätzlich Informationen für die Übertragung auf dem TTEthernet Backbone-Netzwerk notwendig. Diese Informationen werden in der Unterkategorie Backbone definiert. In Kap. 2.5.5 wurden die virtuellen Links erklärt, die TTEthernet zur Adressierung verwendet. Der Eintrag backboneCTID/directMacAdress spezifiziert einen solchen virtuellen Link, der für die Weiterleitung über das TTEthernet Backbone-Netzwerk verwendet werden soll. Außerdem wird die Nachrichtenklasse entsprechend der gewünschten Zeitanforderungen durch den Parameter backboneTransferType angegeben. Unter messageAccumulation wird definiert, ob für diesen virtuellen Link eine Nachrichten-Akkumulation durchgeführt werden soll. Unter Nachrichten-Akkumulation ist eine zeitlich begrenzte Nachrichtenansammlung zu verstehen, um diese dann gesammelt zu übertragen. Die maximale Verzögerungszeit wird mit der holdUpTime angegeben. Die genaue Funktionsweise der Nachrichten-Akkumulation wird in Kapitel 4.4.3 beschrieben.

Der Bereich „gatewayOptions“ ist für weitere Konfigurationen von Gateway-Funktionen vorgesehen. Der Parameter allwaysSendOldAndNew spezifiziert in diesem Falle, ob nur neu eingetroffene Nachrichten oder alte und neue in einer festen Struktur übertragen werden sollen. Besonders strategische Konfigurationen des Gateways können hier untergebracht werden.

Ein Eintrag in der Routing-Tabelle besteht aus einem source-Element, beliebig vielen destination-Elementen und einem gatewayOptions-Element. Für jedes destination-Element kann ein Backbone-Element definiert werden, es muss aber keins angegeben werden. Die Backbone-Informationen sind überflüssig, wenn die Nachricht an ein anderes Bussystem weitergeleitet werden soll, das am selben Gateway angeschlossen ist. Um auch alternative Routingpfade für dieselbe Nachricht definieren zu können, ist die Angabe von mehreren Backbone-Elementen möglich. Durch redundante Übertragung auf unterschiedlichen Pfaden kann die Ausfallsicherheit erhöht werden. Die Filterung der redundanten Nachrichten muss in den Anwendungen der Teilnehmer erfolgen.

Durch die Anordnung der Routing-Tabelle ist es möglich, für ein source-Element sämtliche Routen im TTEthernet Backbone-Netzwerk anzugeben, da beliebig viele destination-Elemente definiert werden können. Zusätzlich können in der Routing-Tabelle am selben Gateway angeschlossene Bus-Systeme referenziert werden, indem ein destination-Element ohne Backbone-Element erstellt wird.

Für jede Nachricht, die in den Gateways verarbeitet werden soll, wird ein Eintrag in der Routing-Tabelle erstellt. Sie kann einmalig global für das gesamte Netzwerk erstellt werden und direkt an alle Gateways übertragen werden. Zur Steigerung der Performance im Gateway können die Einträge gefiltert werden, sodass lediglich Einträge betrachtet werden, in denen entweder sourceBusID oder destinationBusID der eigenen busID entsprechen.

### 4.3.2 Routing Strategie

Der Parameter destinationObjectID aus der Routing-Tabelle definiert den Nachrichten-Identifier für den Destination-Bus. Da nicht davon ausgegangen werden kann , dass die Nachricht im Destination-Bus dieselbe ObjectID haben soll wie im Source-Bus, wird die sourceObjectID in der Nachricht mit der destinationObjectID überschrieben. Dadurch wird eine vollständige Kompatibilität zwischen den Bussystemen und deren Identifier hergestellt. Die Routing-Strategien definieren, ob der Überschreibungsprozess im Source- oder im Destination-Gateway durchgeführt wird.

**PreTransform Routing-Strategie**

Die PreTransform-Routing Strategie beinhaltet, dass bereits vor dem Routing der oben skizzierte Überschreibungsprozess durchgeführt wird. Damit entfällt dieser Schritt im Destination-Gateway. Allerdings kann die Nachricht auch nur für einen Destination-Bus verwendet werden. Diese Strategie ist bei mehreren Empfängern ungeeignet, da in den meisten Fällen für jeden Empfänger sowohl ein virtueller Link als auch eine eigene Nachricht erstellt werden muss.

**PostTransform-Routing Strategie**

Die PostTransform-Routing Strategie schreibt vor, dass die endgültige Zuordnung der destinationObjectID erst im Destination-Gateway erfolgt. Der Überschreibungsprozess

findet demnach erst im Destination-Gateway statt. Eine Nachricht kann auf einem virtuellen Link versendet werden, welcher an unterschiedlichen Destination-Gateways endet.

Bei der PostTransform-Routing-Strategie fallen der Konfigurationsaufwand als auch die Netzwerklast geringer aus als bei der PreTransform Routing-Strategie. Aus diesem Grund wird für das in dieser Arbeit entwickelte Simulationsmodell die PostTransform-Routing Strategie verwendet.

## 4.4 Transformation der Nachrichten

Unter Transformation von Nachrichten ist die Konvertierung von einer Nachrichtendarstellung in eine andere zu verstehen. Im Gateway werden verschiedene Netzwerke mit unterschiedlichen Nachrichtendarstellungen miteinander verbunden, sodass eine Transformation der Nachrichten notwendig wird.

In der Arbeit von Depke (2013) ist ein Ansatz zur Transformation von Nachrichten dargestellt, der eine Transformation in eine einheitliche Transportdarstellung vorsieht, welche dann in eine beliebige Nachrichtendarstellung transformiert werden kann. Für die Übertragung über das TTEthernet Backbone-Netzwerk wird eine Transportdarstellung benötigt, die bei Ankunft an einem oder mehreren Destination-Gateways in die benötigte Nachrichtendarstellung transformiert werden kann. Um eine einheitliche und strukturierte Transformation zu gewährleisten, wird ferner in Depke (2013) mit Protokollschichten gearbeitet. Dadurch kann der Transformationsvorgang sinnvoll strukturiert werden und es lassen sich komponentenbasierte Transformationsarchitekturen entwickeln. Für die Datendarstellung werden Protokollfeldsequenzen verwendet, welche die einzelnen Protokollfelder enthalten. In dieser Arbeit wurde für die Protokollfeldsequenzen ein dynamischer Ansatz entwickelt, der von der strikten Anordnung, wie in Depke (2013) beschrieben, abweicht. Außerdem wurden Nachrichtenfilterung und Zusatzoptionen, die in Depke (2013) durch Konditionale beschrieben werden, vollständig durch Informationen aus der erweiterten Routing-Tabelle ersetzt.

### 4.4.1 Transformationsprotokoll

Das Transformationsprotokoll beschreibt die schrittweise Konvertierung einer Nachrichtendarstellung durch mehrere Protokollschichten. Diese Protokollschichten und die Interaktion untereinander werden nun detailliert dargestellt.

Wie schon erwähnt, sieht das Transformationsprotokoll eine zweistufige Transformation vor. In der ersten Stufe wird eine Nachricht aus einem Bussystem im Source-Gateway in eine einheitliche Transportdarstellung gebracht. Daraufhin werden die Daten über das TTEthernet Backbone-Netzwerk zu einem oder mehreren Destination-Gateways

übermittelt. Im Destination-Gateway findet daraufhin eine Transformation der Transportdarstellung in die benötigte Nachrichtendarstellung des Ziel-Buses statt.

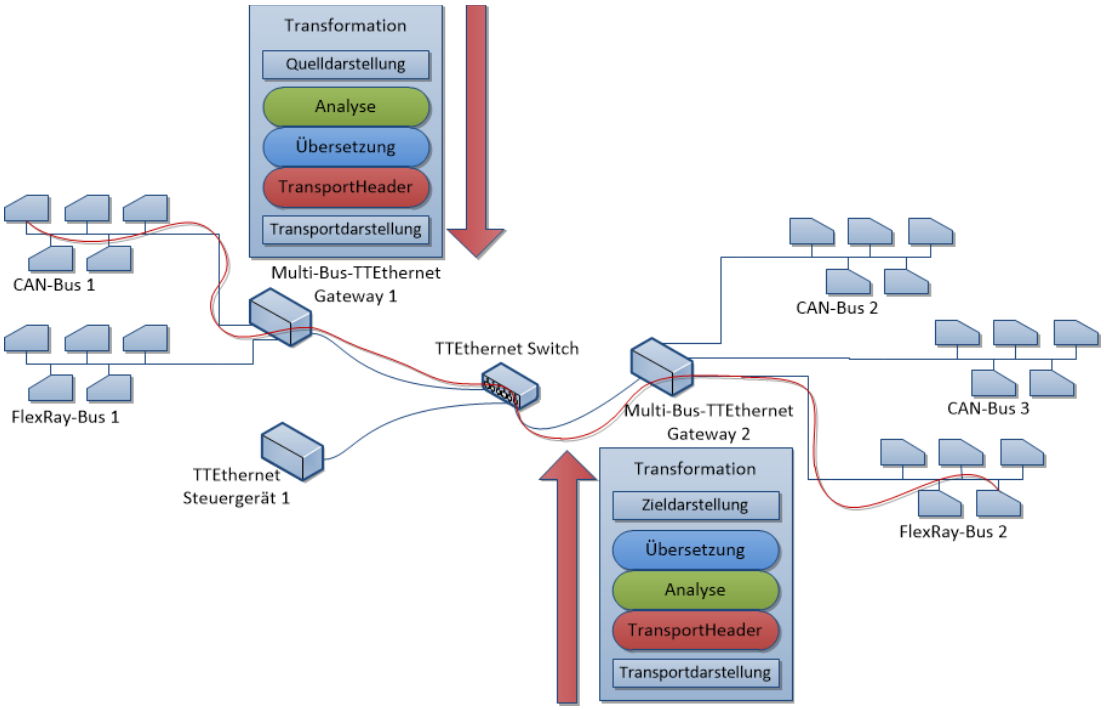


Abbildung XVI: Datenfluss des Transformationsprotokolls

In Abbildung XVI ist die Arbeitsweise des Transformationsprotokolls dargestellt. Sie soll den Datenfluss des Transformationsprotokolls anhand einer Anwendung verdeutlichen. Ein CAN-Frame aus CAN-Bus 1 wird in dem Multi-Bus-TTEthernet Gateway 1 in einen Transportframe konvertiert. Dazu durchläuft der CAN-Frame die Protokollschichten Analyse, Übersetzung und TransportHeader, die weiter unten im Detail erklärt werden. In den Nutzdaten eines TTEthernetframes wird der Transportframe über das TTEthernet Backbone-Netzwerk zu dem Destination-Gateway, das Multi-Bus-TTEthernet Gateway 2, geleitet. Die Informationen für die Weiterleitung über das TTEthernet Backbone-Netzwerk sind in der Routing-Tabelle gespeichert. Im Destination-Gateway durchläuft der Transportframe die Protokollschichten in der Reihenfolge TransportHeader, Analyse und Übersetzung. Das Produkt der Übersetzung-Protokollschicht ist ein Nachrichtenframe in der Zieldarstellung, in diesem Fall ein FlexRay-Frame. Die einzelnen Protokollschichten werden im Folgenden beschrieben.

**Analyse-Protokollschicht**

Die Analyse-Protokollschicht hat die Aufgabe die Semantik der Quell-Darstellung zu ermitteln. Dem Analyse-Algorithmus ist bekannt, bei welcher Frame-Semantik welche Felder für die Transformation relevant sind. In den meisten Fällen werden beispielsweise

Teilnehmer- bzw. ObjektID und die Nutzdaten interessant sein. Diese Felder werden im Folgenden als Protokollfelder bezeichnet. Eine Folge von mehreren Protokollfeldern wird Protokollfeldsequenz genannt. Die Analyse-Protokollschicht schließt das Extrahieren der relevanten Protokollfelder aus der Quell-Darstellung und die Neuordnung in einer Protokollfeldsequenz mit ein. Eine Quell-Darstellung kann ein Bus-Frame oder ein Transportframe sein, je nachdem, ob die Analyse gerade aus Sicht des Source- oder Destination-Gateway betrachtet wird.

**Übersetzungs-Protokollschicht**
Die Übersetzungs-Protokollschicht setzt auf die Analyse-Protokollschicht auf. Sie übergibt der Übersetzungs-Protokollschicht die erstellte Protokollfeldsequenz, die alle relevanten Protokollfelder der Quell-Darstellung beinhaltet. Die Übersetzungs-Protokollschicht bearbeitet die einzelnen Protokollfelder in dem Sinne, dass Werte ausgetauscht oder erweitert werden und gegebenenfalls zusätzliche Protokollfelder hinzugefügt werden. Beispielsweise kann eine CAN-ID im ursprünglichen Bus eine andere als die im Ziel-Bus sein. Zusätzliche Protokollfelder sind Zeitstempel zur Feststellung der tatsächlichen Sendezeit oder Protokollfelder zur Fehlererkennung.

**TransportHeader-Protokollschicht**
Der TransportHeader wird der im Übersetzungsprotokoll erzeugten Protokollfeldsequenz vorangestellt. Dort sind Zusatzinformation zur Herkunft und Transformation der ursprünglichen Nachricht enthalten. Eine sourceBusID verweist auf den Bus der ursprünglich eingegangenen Nachricht. Zusammen mit der Teilnehmer- bzw. Objekt ID einer Nachricht lässt sich eine eindeutige Identifizierung der vorliegenden Daten durchführen.

### 4.4.2 Exemplarische Darstellung am Beispiel eines CAN-Frames

Um den Ablauf und Datenfluss der Transformation zu verdeutlichen, wird der Transformationsvorgang in diesem Abschnitt anhand der Transformation eines CAN-Frames gezeigt. Wie in Kapitel 4.3 zum Thema Routing beschrieben, wird auch in diesem Beispiel nach der PostTransform-Routing Strategie gearbeitet.

Wie in Abbildung XVI dargestellt, haben wir als Ausgangssituation an dem Source-Gateway (Multi-Bus-TTEthernet Gateway 1) einen eintreffenden CAN-Frame. Die Quelldarstellung ist ein CAN-Frame, der nun die Protokollschichten Analyse, Übersetzung und TransportHeader durchläuft.

**Analyse-Protokoll**
Das Analyse-Protokoll erkennt anfangs, dass es sich um einen CAN-Frame handelt. Das bedeutet, dass die CAN-ID, das RTR-Flag und die Nutzdaten extrahiert werden müssen. Diese werden in eine neu angelegte Datenstruktur gespeichert, die das Verhalten einer

Protokollfeldsequenz aufweist. In der folgenden Abbildung XVII ist der Ablauf anhand einer bildlichen Darstellung verdeutlicht.

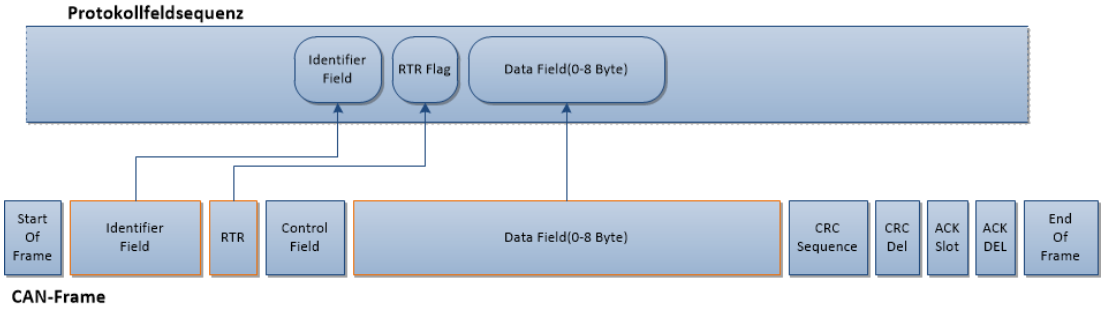


Abbildung XVII: Analyseprotokoll für Can-Frame

### Übersetzungsprotokoll

Die Daten, die im Analyseprotokoll gewonnen wurden, sind in sich vollständig konsistent, sodass diese nicht durch Meta-Informationen gestützt werden müssen. Um den aktuellen Ankunfts-Zeitpunkt des CAN-Frames festzuhalten, wird in das Protokollfeld Timestamp ein Zeitstempel der Ankunftszeit hinzugefügt (s. Abbildung XVIII). Zeitstempel können bei der Sortierung und Plausibilitätsprüfung im Destination-Gateway nützlich sein oder für eine Fehlerdiagnose verwendet werden.

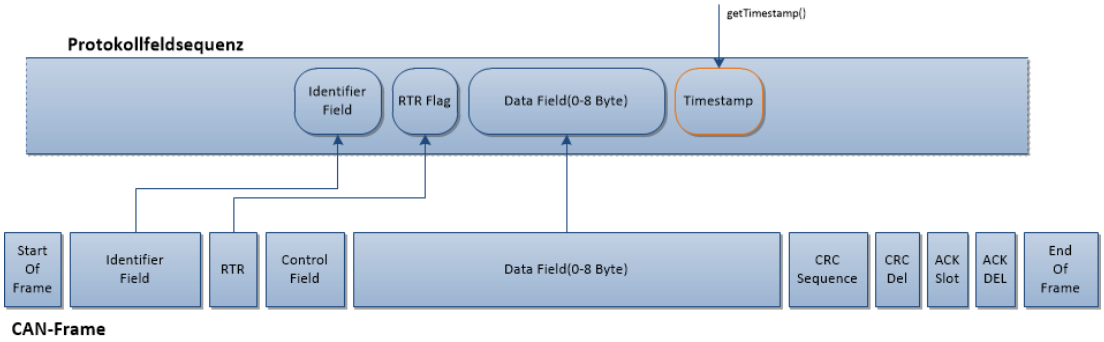


Abbildung XVIII: Übersetzungsprotokoll für CAN-Frame

### Transportheader

Der TransportHeader ergänzt die Protokollfeldsequenz um die sourceBusID. Diese sourceBusID ist für die eindeutige Identifizierung der Einträge in der Routing-Tabelle notwendig. In der folgenden Abbildung wird nun die vollständige Protokollfeldsequenz dargestellt, die nach dem Durchlaufen der drei beschriebenen Protokollschichten als Produkt entsteht. Die vollständige Protokollfeldsequenz wird als Transportframe bezeichnet.

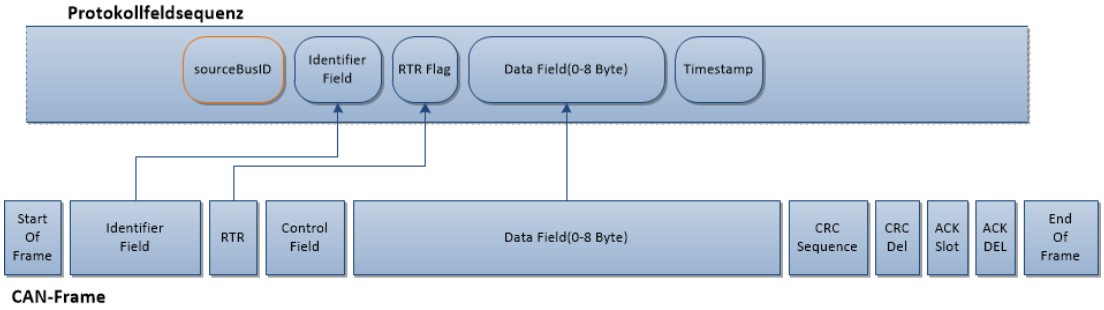


Abbildung XIX: Protokollfeldsequenz mit vollständigem Transportframe eines CAN-Frames

In Abbildung XI befinden wir uns weiterhin im Multi-Bus-TTEthernet Gateway 1 und haben mittlerweile das Transformationsprotokoll durchlaufen, sodass die Transportdarstellung zur Verfügung steht.

**Transport über TTEthernt Backbone-Netzwerk**

Um den Transportframe, welcher der Transportdarstellung entspricht, über das TTEthernet Backbone-Netzwerk zum Destination-Gateway transportieren zu können, wird der Transportframe in die Nutzdaten eines Ethernetframes eingekapselt.

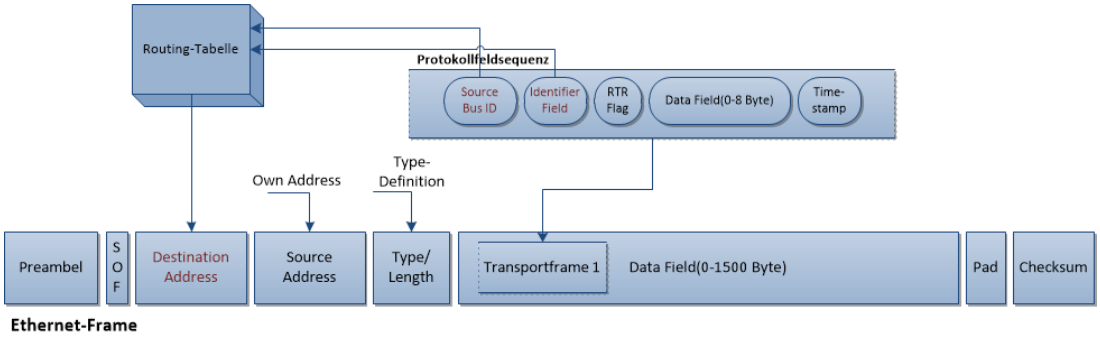


Abbildung XX: Einkapselung des Transportframes in einen Ethernetframe

Abbildung XX zeigt den Ablauf der Einkapselung. Die Ziel-Adresse (Destination Address) ist ein virtueller Link, die durch die sourceBusID und das Identifier Field eindeutig in der Routing-Tabelle bestimmt werden kann. In das Datenfeld (Data Field) wird der Transportframe als eine Einheit eingekapselt. In Kap. 4.4.3 wird ein Konzept zur Einkapselung mehrerer Transportframes erläutert. Die weiteren Protokollfelder werden entsprechend der Ethernet-Semantik verarbeitet.

Der virtuelle Link führt zu dem Multi-Bus-TTEthernet Gateway 2. Dort wird der Transportframe aus dem Ethernetframe entnommen und durch sourceBusID und Identifier Field der entsprechende Eintrag in der Routing-Tabelle gefunden. Dort sind unter anderem die Transformationsinformationen hinterlegt. Im vorliegenden Fall ist die ursprüngliche Darstellung (sourceType) CAN und die Ziel-Darstellung (destinationType) FlexRay. Damit

sind TransportHeader- und Analyse-Protokollschicht im Destination-Gateway durchlaufen. In der Übersetzung-Protokollschicht wird nun das ursprüngliche Identifier Field mit dem Ziel-Parameter(destinationObjectID) ersetzt. Das Datenfeld (Data Field) wird ebenfalls in den FlexRay-Frame kopiert. In Abbildung XXI ist der eben beschriebene Ablauf der Transformation im Destination-Gateway dargestellt.

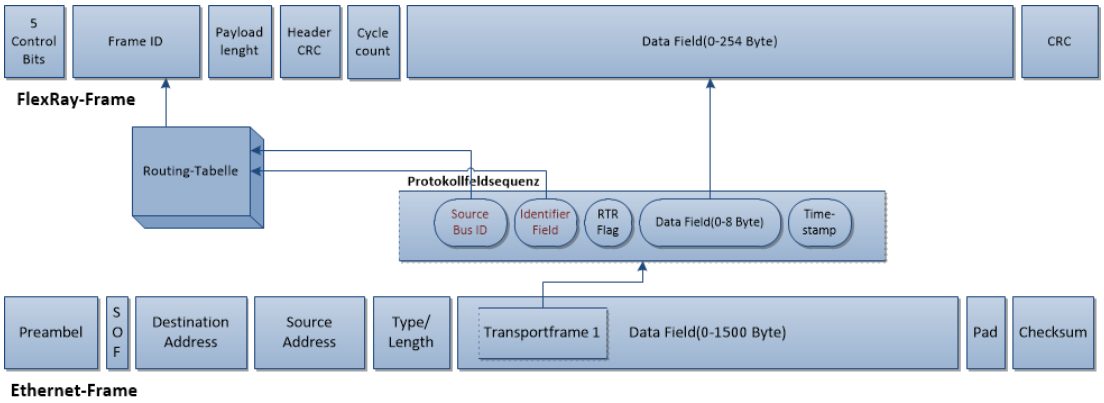


Abbildung XXI: Ablauf Transformation im Destination-Gateway

### 4.4.3 Akkumulation mehrerer Transportframes (Buffering)

Ein Ethernetframe bietet ein weitaus größeres Nutzdaten-Volumen als ein Transportframe in Anspruch nimmt. Das Nutzdaten-Volumen wird stets durch das Pad[22] auf eine minimale Länge von 46 Byte erweitert. Um die Bandbreiten-Auslastung im TTEthernet Backbone-Netzwerk möglichst gering zu halten, sollen mehrere Bus-Frames in einem Ethernetframe zusammen übertragen werden.

Die Akkumulation von mehreren Bus-Frames zieht immer eine Verzögerung im Gateway mit sich, da das Eintreffen mehrerer Nachrichten abgewartet werden muss. Für die Zwischenspeicherung der Nachrichten werden Puffer verwendet, welche im Folgenden Pre-Buffer genannt werden. Jeder virtuelle Link des TTEthernet Backbone-Netzwerkes erhält einen separaten Pre-Buffer. Die Ordnung der Nachrichten im Pre-Buffer erfolgt nach Priorität und nicht nach dem Zeitpunkt des Eintreffens, damit höher priorisierte Nachrichten stets den Vortritt erhalten.

Um die Akkumulation von Nachrichten sinnvoll einsetzen zu können, müssen die virtuellen Links des TTEthernet Backbone-Netzwerkes dem Datenfluss entsprechend festgelegt werden. Viele virtuelle Links führen zu vielen separaten Pre-Buffern, die dann jeweils nur eine entsprechend geringe Anzahl von Nachrichten ansammeln können.

[22] Ethernet-Frame Feld (siehe Abbildung XXI)

Zusätzlich zur überlegten Struktur der virtuellen Links muss eine Strategie für die Art und Länge der Wartezeit im Gateway erstellt werden. Nachfolgend wird eine statische sowie eine dynamische Wartezeitvariante vorgestellt.

**Wartezeitvariante I: statisch**
Auf der Grundlage der vollständigen Ablaufplanung des TTEthernet Backbone-Netzwerkes kann eine optimale Wartezeit für jeden virtuellen Link ermittelt werden. Für diese Berechnung ist generell wichtig, dass das end-to-end delay die Deadline nicht überschreitet. Ausführliche Information ist in (Ayed, et al., 2012)[23] zu finden. Es wird in dieser Arbeit nicht weiter behandelt.

**Wartezeitvariante II: dynamisch**
Für jeden Eintrag in der Routing Tabelle wird eine max. Wartezeit angegeben (siehe Kapitel 4.3). Es ist damit für jeden eintreffenden Bus-Frame eine max. Verzögerungszeit im Gateway bekannt. Bei Weiterleitung eines eintreffenden Bus-Frames an den entsprechenden Pre-Buffer wird der Timer auf das Minimum des aktuellen Timerwertes und der angebenen Verzögerungszeit der eingetroffenen Nachricht gesetzt. Bei Ablauf des Timers wird der aktuelle Inhalt des Pre-Buffers versendet und der Pre-Buffer geleert. Dieser Zyklus startet erneut, sobald der nächste Bus-Frame in den Pre-Buffer hinzugefügt wird.

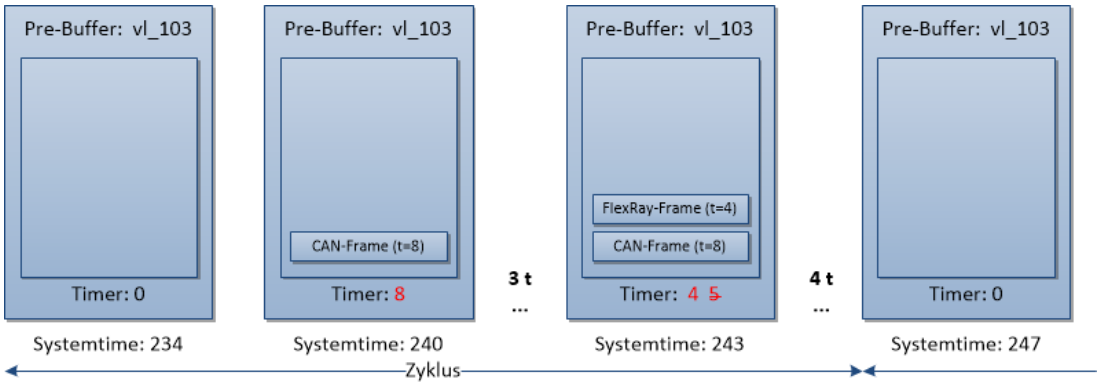


**Abbildung XXII: Darstellung der dynamischen Wartezeitvariante**

Abbildung XXII zeigt das Prinzip der dynamischen Wartezeitvariante. Von links nach rechts ist der zeitliche Verlauf des Pre-Buffers für den virtuellen Link vl_103 dargestellt. Der Zyklus startet mit einem leeren Pre-Buffer und dem Timer im Initialzustand. Der Timer arbeitet nach einem einfachen Zustandsautomaten, welche in Abbildung XXIII dargestellt ist und initial in dem Zustand IDLE wartet. Zur Systemzeit 240 trifft ein CAN-Frame für den virtuellen Link vl_103 ein. Dieser wird im Pre-Buffer zwischengespeichert. Die max. Wartezeit (holdUpTime) für diesen CAN-Frame wurde in der Routing-Tabelle auf 8 Zeiteinheiten spezifiziert. Da dies der erste Frame in diesem Zyklus ist, wechselt der Timer

[23] Frame Packing Strategy within Gateways for Multi-cluster Avionics Embedded Networks

den Zustand von IDLE auf TIMER und setz den Timerwert auf 8 Zeiteinheiten. Zur Systemzeit 243, also 3 Zeiteinheiten später, ist der Timerwert bereits auf 5 dekrementiert. Jetzt trifft ein weiterer Frame für den virtuellen Link vl_103 ein, welcher eine max. Wartezeit von 4 Zeiteinheiten hat. Die Nachrichtendarstellung, in diesem Fall ein FlexRay-Frame, ist dabei nicht von Bedeutung. Der Timer vergleicht nun den aktuellen Timerwert mit dem spezifizierten Timerwert der neuen Nachricht und setzt den Timer auf den kleinsten Wert. Hier wird der Timer auf den Wert der neuen Nachricht mit 4 Zeiteinheiten gesetzt. In diesem Anwendungsbeispiel führt demnach ein Wert < 5 zu einem Timer-Reset, während ein Wert >=5 den Timer unberührt lässt. Nach weiteren 4 Zeiteinheiten ist der Timer abgelaufen. Der gesamte Inhalt des Pre-Buffers wird nun in einer TTEthernet Backbone-Nachricht eingekapselt und verschickt. Nachdem der Pre-Buffer wieder in den Initialzustand gebracht ist, beginnt der Zyklus erneut.

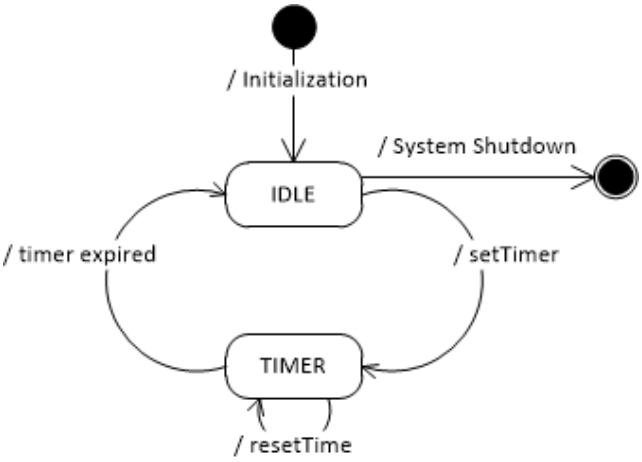


Abbildung XXIII: Zustandsdiagramm des Timers

Um ein adaptives Wartezeitverhalten herzustellen, welches besonders für das eventbasierte Bussystem CAN interessant ist, wird für das Multi-Bus-TTEthernet Gateway die dynamische Wartezeitvariante gewählt.

## 4.5 Anbindung der Bussysteme

Da es keine Synchronisation zwischen den Bussystemen und dem TTEthernet Backbone-Netzwerk auf einer globalen Zeitbasis gibt, arbeiten die Bussysteme nach eigener Taktung. Die Einschleusung von Nachrichten in den Ziel-Bus, die über das TTEthernet Backbone-Netzwerk übertragen worden sind, wird in den nächsten beiden Unterkapiteln für CAN und FlexRay beschrieben.

### 4.5.1 Anbindung von CAN

Da CAN-Nachrichten event-gesteuert versendet werden, kann eine CAN-Nachricht sofort nach Ankunft gesendet werden. Das Arbitrierungsverfahren entscheidet, ob diese Nachricht sofort auf den Bus gelegt werden kann, oder ob weitere Arbitrierungszyklen durchlaufen werden müssen.

### 4.5.2 Anbindung von FlexRay

FlexRay verfügt über ein statisches Segment mit dem Buszugriffsverfahren TDMA und ein dynamisches mit FTDMA[24]. Da auch hier keine festen Zeitangaben zur Ankunft eines Bus-Frames aus Richtung des TTEthernet Backbone-Netzwerks gegeben werden können, erfolgt die Einschleusung von Nachrichten aus Richtung des TTEthernet Backbone-Netzwerks über das dynamische Segment des FlexRay-Bussystems. Eine sinnvolle Priorisierung muss durch den Netzwerk-Designer sichergestellt werden. Der Zyklus-Zähler muss im Destination-Gateway auf den aktuellen Zyklus des Ziel-FlexRay-Bussystems gesetzt werden, da keine Übereinstimmung mit dem Zyklus-Zähler aus der ursprünglichen FlexRay-Nachricht des Quell-FlexRay-Bussystems vorzufinden ist.

[24] FTDMA (Flexible Time Division Multiple Access)

# 5 Simulationsmodell

Dieses Kapitel beschreibt die Umsetzung des Simulationsmodells. In diesem Kontext versteht man unter Simulationsmodell ein Software-Modell, welches bestimmte Eigenschaften der Realität nachbildet. Mit einem Simulationsmodell oder auch einem Zusammenschluss mehrerer Simulationsmodelle, was speziell bei verteilten Systemen interessant ist, können Simulationsabläufe mit unterschiedlichen Konfigurationen erstellt werden, wodurch die Funktionsfähigkeit gezielt, aber auch variabel getestet und analysiert werden kann.

Generell unterscheidet man zwischen kontinuierlichen und diskreten Simulationsmodellen, wobei auch hybride Konstruktionen beider Arten der Simulation möglich sind. Kontinuierliche Simulationen haben einen fest vorgegebenen Verlauf, während diskrete Simulationen ein sprunghaftes Ablaufen von Ereignissen zulassen.

Das Simulationsmodell des Multi-Bus Realtime Ethernet Gateways setzt auf bestehende Simulationsmodelle auf, die in dem objekt-orientierten, modularen, diskreten event-basierten Netzwerk Simulations-Framework OMNeT++ entwickelt worden sind. Aus diesem Grund wurde auch das Simulationsmodell des Multi-Bus Realtime Ethernet Gateways in OMNeT++ entwickelt. In dieser Arbeit wurde das Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways mit Anbindung mehrerer CAN-Bussysteme entwickelt. Eine Erweiterung auf weitere Bussysteme ist vorbereitet.

## 5.1 Einführung in Simulationsumgebung OMNeT++

Die Simulationsumgebung OMNeT++ ist eine lizenzfreie Version für akademische und nicht-kommerzielle Zwecke. Die kommerzielle Version heißt OMNEST[25] und wird unter anderem von CISCO, EADS, Fraunhofer und IBM verwendet.

OMNeT++ simuliert nach der diskreten Simulationsmethode und ist auf Netzwerk-Simulationen spezialisiert. OMNeT++ selbst bietet weniger die Möglichkeiten zu konkreten Simulationen, sondern stellt vielmehr eine Infrastruktur und Tools zur Verfügung, mit denen Simulationen erstellt werden können. Zur Infrastruktur gehört eine Library von Modulen, die durch eigene Implementierungen erweitert werden kann. Die Implementierung des Verhaltens der Module und der Algorithmen erfolgt in C++. Die

[25] OMNEST (Simulcraft Inc.)

einzelnen Module interagieren mit Hilfe von Message-Passing. Für die Modellierung der Netzwerktopologie wird eine OMNeT++ eigene Sprache, die NED[26]-Language, verwendet. Die Konfiguration der Simulationen erfolgt durch ini-Files oder durch Einlesen und Verarbeiten externer Datenquellen - beispielsweise XML.

Für die Simulation bietet OMNeT++ viele Möglichkeiten um Statistiken zu erstellen und zu analysieren. Die Simulation kann außerdem graphisch bis ins Detail dargestellt werden.

### 5.1.1 Module-Struktur in OMNeT++

Eine OMNeT++ Simulation besteht aus hierarchisch vererbten Modulen, die über Nachrichten (Messages) kommunizieren (vgl. OMNeT++ Manual). Module, die ein Verhalten implementieren, werden als Simple Modules bezeichnet. Sie werden in C++ verfasst und erben von Klassen der OMNeT++ Library. Compound Modules vereinen mehrere Module in einer Einheit. Sie implementieren selbst kein Verhalten, sondern führen das Verhalten unterschiedlicher Module zusammen. Sowohl Simple Modules als auch Compound Modules definieren Gates, über welche kommuniziert werden kann. Diese können über Connections miteinander verbunden werden. Es ist auch möglich Gates zu definieren, an die direkt gesendet werden kann, ohne dass die Definition einer Connection erforderlich ist.

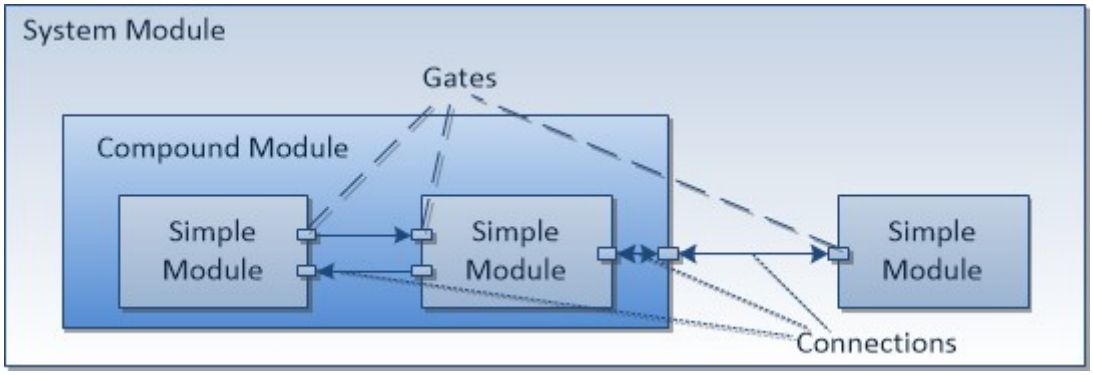


Abbildung XXIV: Modul-Topologie

In Abbildung XXIV sind die eben beschriebenen Komponenten im Zusammenschluss dargestellt. Sowohl System Module, Compound Module als auch Simple Module werden in der NED-Language modelliert und mit Gates und Connections bestückt. Nur Simple Modules haben zusätzliche C++ Files, die Algorithmen oder Anwendungs-Verhalten implementieren.

[26] **NE**twork **D**escription

### 5.1.2 NED-Language

In diesem Abschnitt wird anhand eines Beispiels die NED-Language etwas genauer erklärt. Die folgenden Code-Zeilen beschreiben ein Compound-Module, das den Namen MessageDispatcher hat:

```
module MessageDispatcher
{
   parameters:
      int totalNumberOfVlinks = default(0);
   gates:
      inout moduleConnect @label(InterConnectMsg);
   submodules:
      messageBuffers[totalNumberOfVlinks]: TimeTriggeredBuffer {}
      dispatcher: Dispatcher {}
   connections:
      moduleConnect <--> dispatcher.moduleConnect;
}
```

Dies ist nur ein kleiner Einblick in die NED-Language und soll ein Einblick geben. Mit dem Schlüsselwort **module** wird ein Compound Modul eingeleitet. Ein Simple Module würde mit **simple** eingeleitet werden. Die Bereiche **parameters** und **gates** können sowohl bei Simple als auch Compound Module definiert werden. Die Bereiche **submodules** und **connections** gelten nur für Compound Module. In dem Bereich parameters können Parameter definiert werden, die erst bei Initialisierung der Simulation geladen werden. In den meisten Fällen werden die Werte der Parameter aus ini-Dateien ausgelesen. Auch die Angabe von Default-Werten ist möglich. Der Bereich gates definiert Schnittstellen, an denen andere Simple oder Compound Module angeschlossen werden können. In dem Bereich connections werden die gates verbunden. Unter submodules werden die Unter-Module definiert, die sowohl Simple als auch Compound Module sein können und in dem Compound Modul zusammengeführt werden. Die Verbindungen im Bereich connections können zwischen dem umschließenden Compound Modul und dessen Unter-Modulen bestehen, aber auch zwischen den Unter-Modulen selbst sein. In allen Bereichen kann mit Arrays gearbeitet werden, wie im Beispiel bei messageBuffers im Bereich submodules.

### 5.1.3 Messages

Nachrichten (Messages) sind in OMNeT++ sehr wichtig und werden vielseitig verwendet. Nachfolgend wird auf die Grundfunktionalität, nämlich den Datenaustausch zwischen Modulen, eingegangen. Die Basisklasse aller Messages ist die cMessage-Klasse. Sie folgt mit der OMNeT++ Library mit und enthält wichtige Strukturen und Parameter für den Nachrichtenaustausch. Für die Simulation des Netzwerkverkehrs gibt es eine spezialisierte Klasse, die den Namen cPacket trägt. Sie enthält vor allem die Verwaltung der Paketgröße

und bietet die Möglichkeit, cPacket-Objekte ineinander zu verschachteln, um Netzwerkprotokollschichten simulieren zu können.

In den meisten Fällen werden allerdings spezielle Parameter bzw. Protokollfelder benötigt. Durch eine sehr einfache Message-Definition lassen sich auf Basis von cMessage und cPacket automatisch spezialisierte Message-Typen generieren. Reicht die automatische Generierung für die gewünschte Anwendung nicht aus, so ist eine weitere Anpassung in C++ möglich.

## 5.2 Architektur des Gateway Simulationsmodells

In dieser Arbeit wurde ein Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways entwickelt. Im Folgendem wird eine Übersicht über die Komponenten des Gateways gegeben und die einzelnen Komponenten detailliert erklärt.

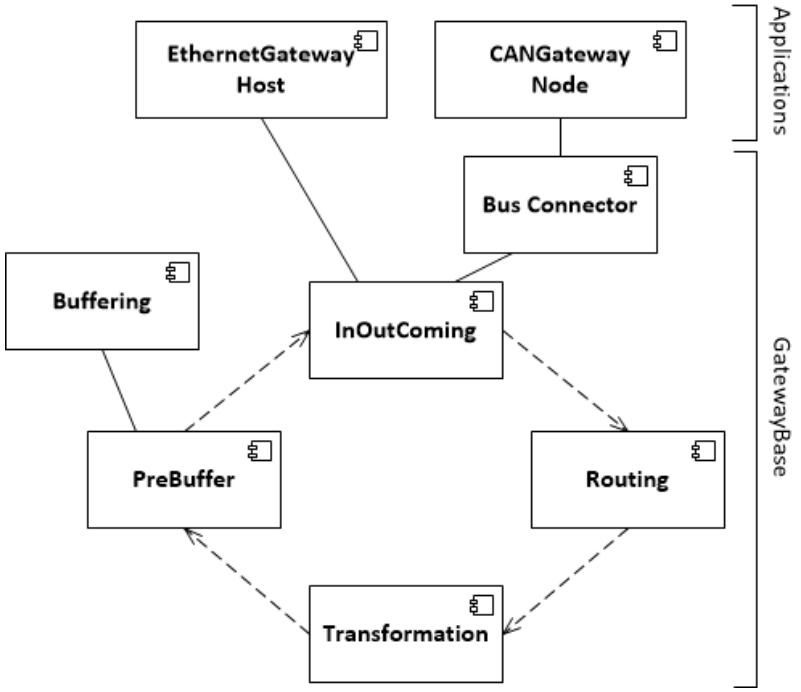


Abbildung XXV: Komponentendiagramm des Gateways

Das Gateway bildet das Bindungsglied zwischen dem TTEthernet Backbone-Netzwerk und den Bussystemen. Im Simulationsmodell wurde im ersten Schritt eine Anbindung mehrerer CAN-Bussysteme implementiert und einfache Erweiterungsmöglichkeiten auf weitere Bussysteme vorbereitet. Das Simulationsmodell fasst alle in Abbildung XXV gezeigten Module in einem Compound Modul BusEthernetGateway zusammen (Siehe Abbildung XXVI).

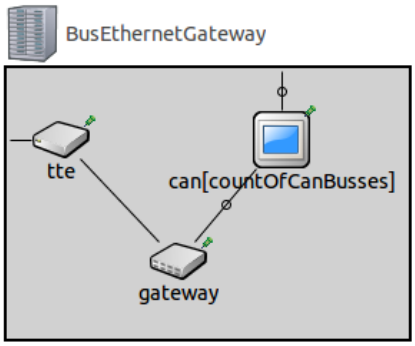


Abbildung XXVI: BusEthernetGateway

Das Modul tte steht für die Anbindung an das TTEthernet Backbone-Netzwerk, das durch das Compound Modul EthernetGatewayHost repräsentiert wird. Das Modul can steht für die Anbindung mehrerer CAN-Bussysteme und wird durch ein oder mehrere Compound Module repräsentiert, die man CanGatewayNode nennt. Das Modul gateway (Compound Module: GatewayBase) beinhaltet die Gateway-Logik und besteht aus den Komponenten InOutComing (mit BusConnector), Routing, Transformation und PreBuffer (mit Buffering). Auf die einzelnen Teilbereiche wird später detailliert eingegangen.

Das Komponentendiagramm des Gateways (Abbildung XXV) illustriert, dass Nachrichten (Messages) zwischen Applikationen und GatewayBase in beiden Richtungen ausgetauscht werden können. Nachrichten, die von einem der Applikations-Module stammen, durchlaufen den Kreislauf der Gateway-Komponenten Routing, Transformation und PreBuffer. In den meisten Fällen wird eine Nachricht nach dem Passieren des Kreislaufes durch die InOutComing-Komponente an die komplementäre Applikation zur weiteren Verarbeitung geleitet, da gerade darin die Aufgabe des Gateways besteht. Allerdings kann auch der Fall eintreten, dass eine Nachricht eines CAN-Bussystems an ein CAN-Bussystem weitergeleitet werden soll, das am selben Gateway angeschlossen ist. In diesem Fall stammt die Nachricht von der Applikation der Bus Anbindung und wird nach dem Passieren des Kreislaufes wieder an diese Applikation geleitet.

### 5.2.1 Datenfluss und Nachrichten-Darstellungen

Um die Funktion des Gateways verständlicher darzustellen, werden zunächst der Datenfluss und die Nachrichten-Darstellung beschrieben. In Abbildung XXVII ist der Ablauf vom Eintreffen eines CAN-Frames am Source-Gateway bis zur Weiterleitung auf den Destination-Bus im Destination-Gateway dargestellt, wobei die Nachrichten-Akkumulation außen vor gelassen wurde. Die blauen Elemente zeigen die Nachrichten-Darstellung des Source-Gateways und die roten Elemente die Nachrichten-Darstellung des Destination-Gateways.

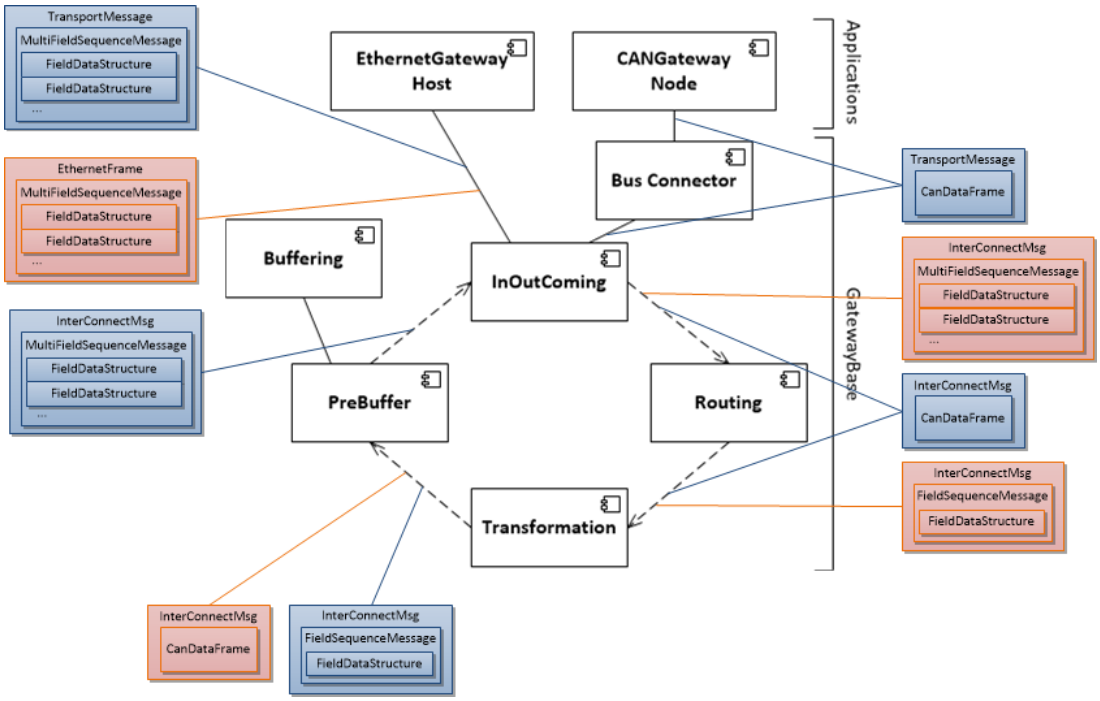


**Abbildung XXVII: Datenfluss mit Nachrichten-Darstellung**

Im Source-Gateway wird ein CanDataFrame von CANGatewayNode bis zum InOutComing in einer TransportMessage übermittelt. Bis zur Transformation wird der CanDataFrame in einer InterConnectMsg transportiert. Die Transformation des CanDataFrames führt zu einer Transport-Darstellung, welche FieldSequenceDataStructure genannt wird. Diese wird zur weiteren Verarbeitung in eine FieldSequenceMessage eingebunden und in der InterConnectMsg an das PreBuffer Modul geleitet. Nach der Pufferung werden mehrere FieldSequenceDataStructures (Transport-Darstellung von CanDataFrames) in einer MultiFieldSequenceMessage zusammengefasst und in der InterConnectMsg an InOutComing weitergeleitet. Die MultiFieldSequenceMessage geht in einer TransportMessage an den EthernetGatewayHost.

Im Destination-Gateway kommt diese MultiFieldSequenceMessage in einem EthernetFrame eingekapselt an der Komponente InOutComing an. Dort wird sie als InterConnectMsg an Routing weitergeleitet. Das Routing-Modul splittet die MultiFieldSequenceMessage auf, sodass für jede enthaltene FieldSequenceDataStructure eine FieldSequenceMessage generiert und in einer InterConnectMsg weitergeleitet wird. Dies ist für die korrekte Verarbeitung der FieldSequenceDataStructure in dem Transformation-Modul notwendig. Nach der Transformation steht der CanDataFrame für das Destination-Bussystem zur Verfügung. Dieser CanDataFrame wird, eingekapselt in eine InterConnectMsg, über das PreBuffer-Modul direkt an InOutComing weitergeleitet. Dort nimmt der CanDataFrame den Weg bis zum CANGatewayNode in einer TransportMessage.

**FieldSequenceDataStructure**
Der FieldSequenceDataStructure kommt eine besondere Bedeutung zu, weil sie das Prinzip der Protokollfeldsequenz darstellt. Sie bietet generische Funktionen zum Hinzufügen und Auslesen von Elementen. Alle enthaltenen Elemente müssen von einer Basis-Klasse FieldElement erben. Welche, wie viele und in welcher Reihenfolge Elemente hinzugefügt werden spielt keine Rolle. Das Auslesen von Elementen ist durch die Angabe des Element-Typs möglich. In Kapitel 5.2.7 (Abbildung XXXIII) ist eine tabellarische Übersicht der genutzten FieldElements dargestellt.

### 5.2.2 Global-Gateway-Information Registrierung/ automatische Konfig.

Für alle Gateways gibt es Meta-Informationen, die für alle Komponenten innerhalb eines Gateways verfügbar sein müssen. Dies sind Informationen darüber, welche Bussysteme angeschlossen sind und welche Puffer für die Nachrichten-Akkumulation zur Verfügung stehen. Das Modul Global-Gateway-Information ist die globale Instanz, die diese Informationen verwaltet. Es enthält statische Funktionen, durch welche diese Informationen registriert werden können.

In einer Initialisierungsphase der Simulation findet eine automatische Konfiguration und Registrierung statt. Im ersten Schritt registriert sich das Gateway mit dem Gateway-Namen bei dem Global-Gateway-Information Modul. Daraufhin können die Bussysteme unter Angabe des spezifischen Gateways registriert werden. Außerdem werden die im Gateway notwendigen Puffer für die Nachrichten-Akkumulation (TimeTriggedBuffer) registriert. Diese Registrierung erfolgt ebenfalls mit Angabe eines spezifischen Gateways.

Aus der Routing-Tabelle sind die virtuellen Links bzw. Mac-Adressen bekannt, die von einem Gateway genutzt werden. Auf Grundlage dieser Informationen aus der Routing-Tabelle werden die TimeTriggeredBuffer automatisch generiert, konfiguriert und bei dem Global-Gateway-Information Modul registriert.

Nach der Initialisierungsphase ist der Registrierungsvorgang abgeschlossen, da alle notwendigen Informationen vorliegen. Auf diese Weise können während der Simulation sämtliche Informationen separat für jedes einzelne Gateway abgefragt werden. Das Global-Gateway-Information Modul bietet viele Funktionen, die die Handhabung der Informationen erleichtern.

Für die Simulation wurde der Ansatz der globalen Verwaltung gewählt, um die Registrierung während der Initialisierung auf einfache Weise durchzuführen. Für reale Systeme ist dagegen eine interne Verwaltung und Registrierung innerhalb eines Gateways vorzuziehen.

### 5.2.3 Anbindung des EthernetGatewayHosts am Gateway

Der TTEthernet Connector (EthernetGatewayHost) im Gateway ist durch einen Netzwerkknoten des Time-Triggered Ethernet realisiert, welcher ein gateway-spezifisches Verhalten implementiert. In den nächsten beiden Abschnitten wird zum einen unter Physical Layer ein Einblick in den Aufbau des Netzwerkknotens gegeben und zum anderen unter Data Link Layer die Anbindung auf Protokollebene dargestellt.

**Physical Layer**

Das Time-Triggered Ethernet agiert als Backbone-Netzwerk. Dies ist ein bereits bestehendes Simulationsmodell mit dem Namen CoRE4INET und wurde an der HAW in der Forschungsgruppe CoRE[27] entwickelt.

Um das Gateway-Verhalten auf einen Netzwerkknoten zu übertragen kann dieser durch ein Applikations-Modul erweitert werden, welches das gewünschte Verhalten implementiert. Um die Konfiguration des Gateways zu erleichtern, wurden außerdem Vektoren für die virtuellen Links angelegt.

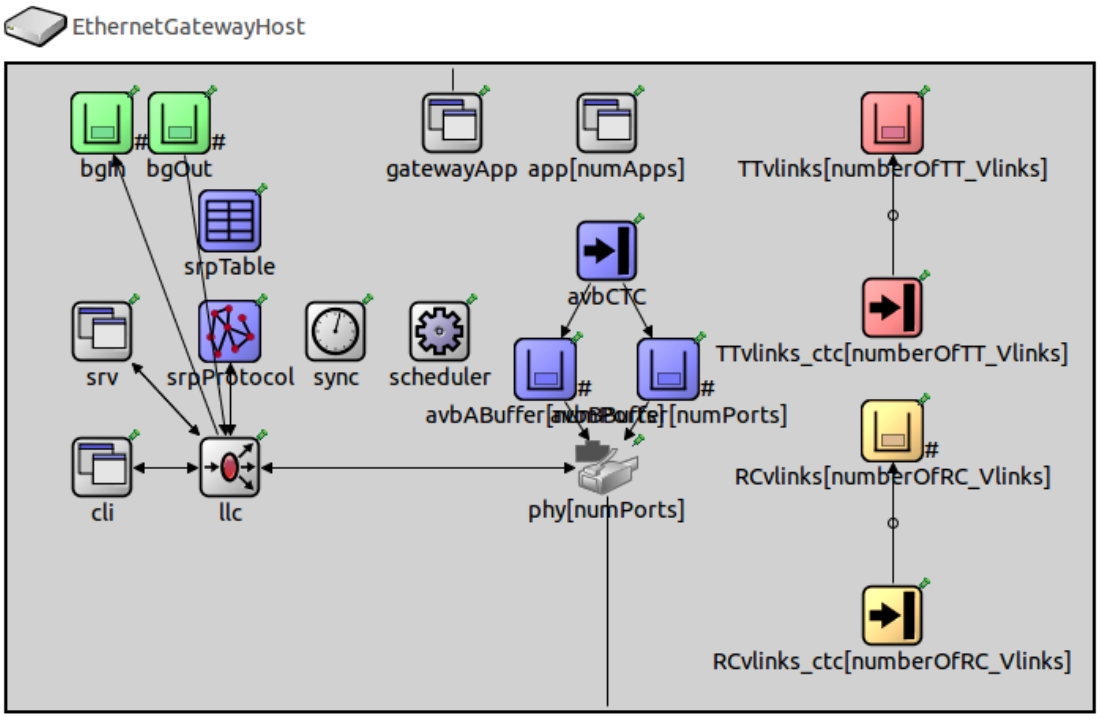


Abbildung XXVIII: EthernetGatewayHost

In Abbildung XXVIII ist die graphische Ansicht des Compound Modules EthernetGatewayHost mit all seinen Komponenten dargestellt. Der phy[numPorts] stellt

[27] (CoRE Research Group)

das Ethernet-Interface dar und kann auch für mehrere Ethernet-Anschlüsse konfiguriert werden. Von Bedeutung ist das Modul gatewayApp, welches das Verhalten des Ethernet-Knotens im Gateway implementiert. Dieses Modul hat eine Verbindung nach außen (zu erkennen an der vertikalen Linie), welche zu den Gateway-Modulen führt. Diese werden später detailliert erklärt und sind in dem Compound Module GatewayBase zusammengefasst. Auf der rechten Seite sind die Module jeweils für den Empfang und die Pufferung von TT- und RC-Nachrichten dargestellt. Best-Effort Nachrichten (BE) werden über bgIn und bgOut verwaltet.

**Data Link Layer**

Nachrichten des TTEthernet Backbone-Netzwerkes kommen bei dem Modul phy an und werden an den entsprechen TT-, RC- oder BE-Puffer weitergeleitet. Diese Nachrichten werden durch die gatewayApp weiter verarbeitet, indem sie in eine TransportMessage eingekapselt werden und an die GatewayBase gesendet werden. Die TransportMessage stellt das Nachrichtenformat dar, das zur Kommunikation zwischen den Applikationen und der GatewayBase verwendet wird. Auf diese Weise können sämtliche Nachrichtendarstellungen in einem Standardformat zwischen Applikationen und GatewayBase ausgetauscht werden. Die vollständige Programmlogik der gatewayApp ist in Abbildung XXIX als Nassi-Shneiderman[28]-Diagramm dargestellt.

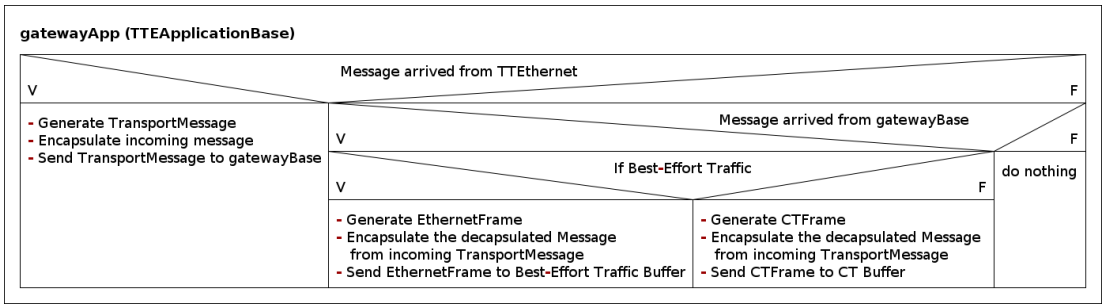


**Abbildung XXIX: Nassi-Shneiderman-Diagramm der gatewayApp**

Die Weiterleitung der Nachrichten aus Richtung des TTEthernet wurde am Anfang des Kapitels Data Link Layer bereits beschrieben. Nachrichten, die aus Richtung der GatewayBase kommen, werden als EthernetFrame oder CTFrame an das TTEthernet Backbone-Netzwerk weitergeleitet. Sofern Best-Effort Traffic in der Routing-Tabelle für diese Nachricht konfiguriert ist, wird die Nachricht als ein Standard EthernetFrame weitergeleitet. Ist TimeTriggered oder Rate-Constraint Traffic konfiguriert, so wird ein CTFrame für die Weiterleitung der Nachricht verwendet.

[28] Diagrammtyp zur Darstellung von Programmentwürfen

### 5.2.4 Anbindung des CANGatewayNodes über den BusConnector am Gateway

Generell können ein oder mehrere CANGatewayNodes angebunden werden. Für jedes CAN-Bussystem, das an das Gateway angebunden werden soll, muss jeweils ein CANGatewayNode im Gateway integriert werden. Die Steuerung und Verwaltung von den CAN-Bussystemen erfolgt durch das Modul BusConnector.

**Funktion des BusConnectors**

Wie in Kapitel 5.2.2 beschrieben, werden die angeschlossenen Bussysteme während der Initialisierung durch den BusConnector global registriert. In diesem Abschnitt wird die Funktion des BusConnectors während der Simulation beschrieben. In Abbildung XXX ist der Programmablauf als Nassi-Shneiderman-Diagramm dargestellt. Für eintreffende Nachrichten aus Richtung des Gateways werden die entsprechenden Destination-Elemente aus der Routing-Tabelle geladen. Dort sind unter anderem Informationen enthalten, welcher Destination-Bus die Nachricht erhalten soll. Ist das entsprechende Bussystem am Gateway registriert, so wird der CanDataFrame in eine TransportMessage eingekapselt und zum entsprechenden CANGatewayNode weitergeleitet. Grundsätzlich sind alle Nachrichten aus Richtung der Bussysteme TransportMessages mit eingekapselten CanDataFrames. Diese werden zur GatewayBase weitergeleitet.

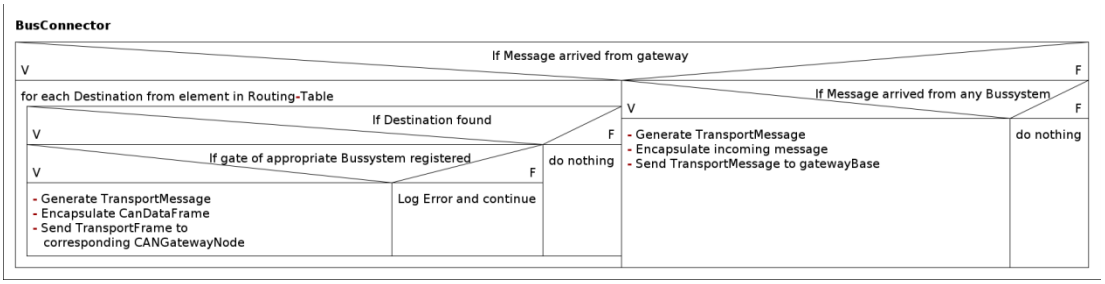


Abbildung XXX: Nassi-Shneiderman-Diagramm des BusConnector

**Physical Layer des CANGatewayNodes**

Wie das Simulationsmodell des Time-Triggered Ethernet wurde auch das Simulationsmodell für CAN an der HAW Hamburg in der Forschungsgruppe CoRE[29] entwickelt. Der CANGatewayNode ist ein normaler CAN-Teilnehmerknoten, der um die Gateway Applikationen erweitert wurde.

[29] (CoRE Research Group)

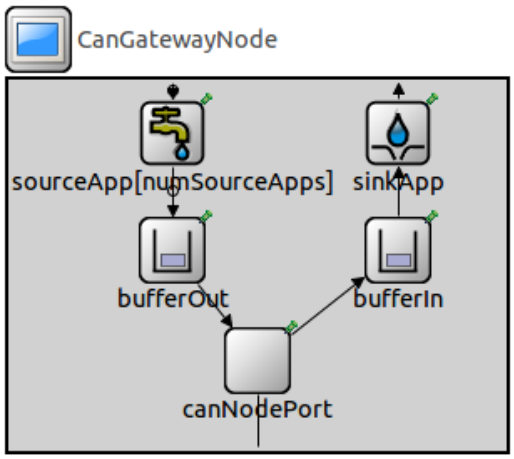


Abbildung XXXI: CANGatewayNode

In Abbildung XXXI ist der CANGatewayNode aus der Simulation dargestellt. Der canNodePort stellt die Verbindung zum Bus sicher. Die Module bufferIn und bufferOut sorgen für die Zwischenspeicherung der Nachrichten. Das Verhalten der Knoten wird durch die sourceApp und sinkApp realisiert. In der sourceApp können CAN-Frames erzeugt und ein Sendewunsch erteilt werden. Die sinkApp nimmt die eingehenden Nachrichten entgegen und verarbeitet sie. Diese Applikationen, sowohl sourceApp als auch sinkApp, wurden durch Applikationen ersetzt, die das Gateway-Verhalten implementieren. Diese haben eine Verbindung zur GatewayBase, sodass das CAN-Bussystem für das Gateway zugänglich ist.

**Data Link Layer des CANGatewayNodes**
Aus Richtung der GatewayBase (siehe Abbildung XXXII) kommen die Nachrichten als TransportMessages in der sourceApp an. Der CAN-Frame wird aus der TransportMessage entnommen und an das Modul bufferOut gesendet. Sobald das Modul bufferOut den Frame zur Versendung frei gibt, wird der CAN-Frame durch den canNodePort auf den Bus gelegt.

Nachrichten aus dem CAN-Bussystem treffen über canNodePort und bufferIn in der sinkApp ein. Diese generiert eine TransportMessage und leitet sie an die GatewayBase zur weiteren Verarbeitung weiter.

### 5.2.5 InOutComing

Das Modul InOutComing ist als Schaltzentrale für die korrekte Weiterleitung und Nachrichten-Darstellung der Nachrichten innerhalb des Gateways verantwortlich. In Abbildung XXXII auf Seite 58 ist es unter der Bezeichnung iocom zu finden. Ohne im Detail auf die vielfältigen Abhängigkeiten in der Weiterleitung eingehen zu wollen, wird hier kurz angedeutet, welche Funktion InOutComing hat. Abbildung XXXIIEs agiert als Bindeglied

zwischen den Kommunikationssystemen und der Gateway-Funktionalität. Die GatewayBase arbeitet zur internen Kommunikation mit der InterConnectMessage, während die Kommunikation mit den angeschlossenen Kommunikationssystemen, Time-Triggered Ethernet und CAN, über die TransportMessage erfolgt. Die Schaltzentrale InOutComing konvertiert die Nachrichten entsprechend dem Datenfluss.

### 5.2.6 Routing

In Kapitel 5.2 wurden die Komponenten des Simulationsmodells dargestellt. Das Routing Modul ist das erste Modul, das eine grundlegende Funktion des Gateways aufweist. Für die Module Routing, Transformation und PreBuffer (Buffering) muss intern zwischen Source- und Destination Gateway unterschieden werden. Anhand der Nachrichten-Darstellung der eintreffenden Nachricht entscheiden die Module, welche Funktion ausgeführt wird.

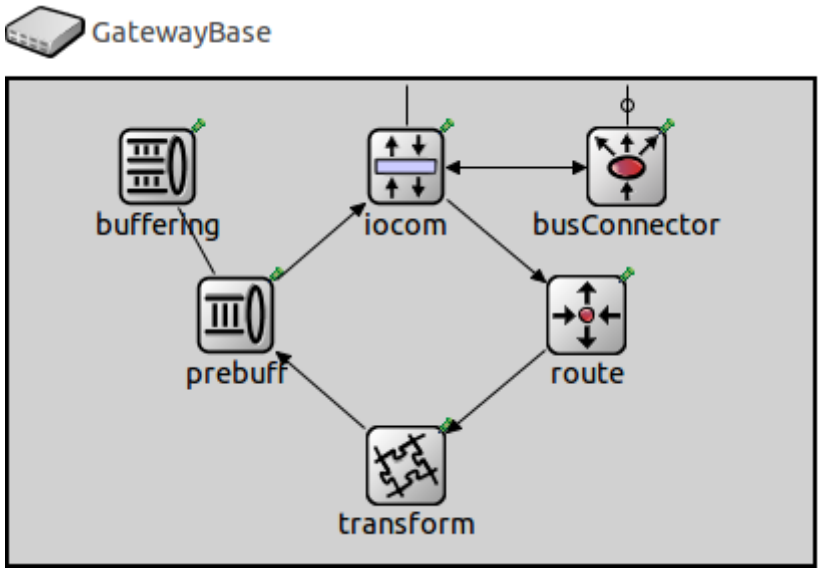


Abbildung XXXII: Modul-Übersicht der GatewayBase

Zur Orientierung ist in Abbildung XXXII die Modul-Übersicht der GatewayBase dargestellt. Das Routing-Modul hat den Namen route.

Die Grundlage für das Routing bildet die Routing-Tabelle. Sie wird als ein XML-File definiert und allen Gateways zur Verfügung gestellt. Eine globale Definition erleichtert die Übersichtlichkeit über die Netzwerk-Konfiguration.

**Funktion als Source-Gateway**

Ein Source-Gateway bekommt CAN-Frames und schickt Ethernet-Nachrichten an das Backbone-Netzwerk. In einigen Fällen kann aber auch eine Nachricht an ein anderes CAN-

Bussystem desselben Gateways gesendet werden, sofern dies in der Routing-Tabelle definiert ist.

Die Routing-Tabelle wird als XML-File definiert und direkt von der Simulationsumgebung OMNeT++ eingelesen. OMNeT++ bietet auch Möglichkeiten zur Verwaltung und Bearbeitung der XML-Daten während der Simulation an.

Die sourceBusID und sourceObjectID sind in der eingegangenen Nachricht enthalten. Anhand dieser Parameter werden die XML-Daten durchsucht und der entsprechende Datensatz in der InterConnectMsg für die weitere Verarbeitung im Gateway referenziert. Daraufhin wird die InterConnectMsg an das Modul Transformation weitergeleitet.

**Funktion als Destination-Gateway**
Ein Destination-Gateway bekommt eine MultiFieldSequenceMessage und gibt die dort enthaltenen Bus-Frames an die Bussysteme weiter. Eine MultiFieldSequenceMessage beinhaltet eine Vielzahl an FieldSequenceDataStructures. Eine FieldSequenceDataStructure repräsentiert einen Transport-Frame, also einen Bus-Frame in Transport-Darstellung.

In dem Routing-Modul werden die FieldSequenceDataStructures aus der MultiFieldSequenceMessage einzeln weiterverarbeitet. Anhand der sourceBusID und der sourceObjectID aus der FieldSequenceDataStructure werden die entsprechenden Routing-Daten geladen und einer InterConnectMsg angefügt. Die FieldSequenceDataStructure wird nun einzeln in einer FieldSequenceMessage weiterverarbeitet, die nur eine FieldSequenceDataStructure enthalten kann. Abschließend wird die InterConnectMsg mit der eingekapselten FieldSequenceMessage an das Modul Transformation weitergeleitet. Dieser Vorgang wird für jede enthaltene FieldSequenceDataStructure der MultiFieldSequenceMessage durchgeführt.

### 5.2.7 Transformation

Das Modul Transformation hat die grundlegende Aufgabe, Bus-Frames in die Transport-Darstellung (FieldSequenceDataStructure) und Transport-Darstellungen in Bus-Frames zu konvertieren. Zusätzlich werden einige administrative Aufgaben zur Vorbereitung der Nachrichten-Akkumulation durchgeführt. Das Transformation-Modul ist unter der Bezeichnung transform in Abbildung XXXII auf Seite 58 dargestellt.

Wie auch das Routing-Modul agiert das Transformation-Modul sowohl als Source- als auch als Destination-Modul. In wenigen Fällen agiert es gleichzeitig als Source- und als Destination-Modul. Dies ist der Fall, wenn ein Bus-Frame direkt an einen anderes Bussystem desselben Gateways geleitet werden soll. Dann wird allerdings keine Transformation, sondern eine direkte Weiterleitung durchgeführt.

Die Transformation kann in diesem Modul für sämtliche Bussysteme durchgeführt werden. Der entsprechende Transformationsalgorithmus wird anhand des source- und destinationTypes aus der Routing-Tabelle ermittelt. Im Simulationsmodell wird zunächst nur das CAN-Bussystem verwendet, dessen Transformation nun detailliert beschrieben wird.

**Funktion als Source-Gateway**
Die Funktion des Transformation-Modules im Source-Gateway besteht darin, Bus-Frames in die Transport-Darstellung zu konvertieren. Die Transport-Darstellung wird durch die FieldSequenceDataStructure realisiert, wie sie in Kapitel 5.2.1 erläutert wurde. Es werden Objekte der einzelnen Protokollfelder erstellt, beispielsweise dem IdentifierFieldElement, welches die CAN-ID enthält. Diese Objekte werden der FieldSequenceDataStructure hinzugefügt.

| Objekt-Typ | Beinhaltet folgende Protokollfelder |
|---|---|
| **TransportHeaderFieldElement** | ID des CAN-Bussystems (sourceBusID) |
| **IdentifierFieldElement** | CAN-ID (sourceObjectID) |
| **DataFieldElement** | Nutzdaten |
| **RTRFieldElement** | RTR-Flag (Remote-Frames) |
| **TimestampFieldElement** | Zeitstempel der Ankunft des CAN-Frames |

**Abbildung XXXIII: Elemente der FieldSequenceDataStructure für einen CAN-Frame**

In Abbildung XXXIII sind alle Elemente aufgelistet, welche in der FieldSequenceDataStructure eines CAN-Frames enthalten sind. Zur weiteren Verarbeitung wird die FieldSequenceDataStructure in einer FieldSequenceMessage verpackt. Diese wird in die InterConnectMsg eingekapselt und an das Modul PreBuffer (Buffering) weitergeleitet.

**Funktion als Destination-Gateway**
Im Destination-Gateway hat das Modul Transformation die Aufgabe, aus der Transport-Darstellung (FieldSequenceDataStructure) einen CAN-Frame zu erstellen. Im Prinzip können alle Elemente der FieldSequenceDataStructure in den CAN-Frame übertragen werden. Allerdings muss die CAN-ID aus der ursprünglichen Nachricht durch die destinationObjectID (CAN-ID für das Destination-Bussystem) ersetzt werden, da nach dem Prinzip der PostTransform-Routing Strategie (Siehe Kap. 4.3.2) verfahren wird. Der auf diese Weise erstellte CAN-Frame wird der InterConnectMsg angefügt und an das Modul PreBuffer weitergeleitet.

### 5.2.8 PreBuffer (Buffering)
In diesem Modul wird die Nachrichten-Akkumulation realisiert. Genauer gesagt handelt es sich um einen Modul-Zusammenschluss, der diese Funktionalität repräsentiert. Das Modul PreBuffer ist die übergeordnete Komponente und steuert die Verteilung, sowie Ein- und

Ausgang für die Funktion der Nachrichten-Akkumulation. Dieses Modul ist in Abbildung XXXII auf Seite 58 mit der Bezeichnung prebuff dargestellt.

**Funktion als Source-Gateway**

Wie in Kapitel 5.2.7 beschrieben, trifft eine Nachricht in dem Modul PreBuffer in der Nachrichten-Darstellung als FieldSequenceMessage ein. Wurde in der Routing-Tabelle für diese Nachricht keine Nachrichten-Akkumulation definiert (MessageAccumulation == false), so wird der Transport-Frame (FieldSequenceDataStructure) aus der FieldSequenceMessage in eine MultiFieldSequenceMessage übertragen und an das Modul InOutComing weitergeleitet. Dies ist erforderlich, da die Daten über das Backbone-Netzwerk eine einheitliche Darstellung als MultiFieldSequenceMessages haben sollen. Dabei spielt es keine Rolle, ob eine oder mehrere Nachrichten in der MultiFieldSequenceMessages enthalten sind.

Die Funktion der Nachrichten-Akkumulation tritt in Kraft, wenn eine Nachricht eintrifft, für die in der Routing-Tabelle das messageAccumulation-Flag auf true gesetzt ist. In diesem Fall leitet das PreBuffer-Modul die FieldSequenceMessage an die Buffering Komponente weiter, die in Abbildung XXXIV dargestellt ist und als MessageDispatcher bezeichnet wird.

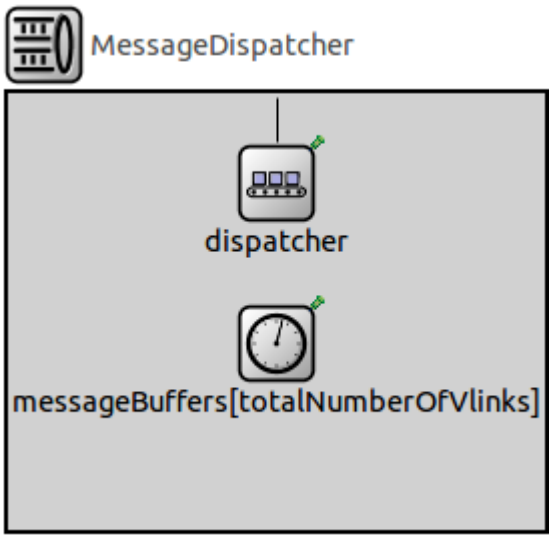


Abbildung XXXIV: Buffering (MessageDispatcher)

Die Nachrichten treffen in dem Modul dispatcher ein. Die Puffer für die Nachrichten-Akkumulation verbergen sich hinter messageBuffers. Jeder virtuelle Link bzw. jede Mac-Adresse erhält einen eigenen Puffer.

In dem Modul Dispatcher wird anhand des virtuellen Links (TT- bzw. RC-Übertragung) oder Mac-Adressen (BE-Übertragung) der entsprechende Puffer identifiziert. Die FieldSequenceMessage wird direkt an diesen Puffer gesendet. Das direkte Versenden von

Messages erfordert keine Connections (Verbindungen). Deshalb sind in Abbildung XXXIV zwischen dispatcher und messageBuffers keine Verbindungen angezeigt.

Hinter den messageBuffers verbergen sich Module, die sich TimeTriggeredBuffer nennen. Die Funktionsweise des TimeTriggeredBuffers ist detailliert in Kapitel 4.4.3 dargestellt. Daher wird es hier nur kurz angedeutet. Die Nachrichten im TimeTriggeredBuffer werden unabhängig von der Ankunftszeit der Priorität nach geordnet, die sich aus der CAN-ID ergibt. Das Prinzip der dynamischen Wartezeit ist durch einen Timer realisiert: Ist der Timer-Wert höher als die maximale Wartezeit einer eintreffenden Nachricht, so wird der Timer-Wert auf den kleineren Wert neu gesetzt. Nach Ablauf des Timers tritt ein Event ein, das abgefangen wird. Daraufhin erzeugt der TimeTriggeredBuffer eine MultiFieldSequenceMessage mit allen Nachrichten, die sich zurzeit im Puffer befinden und sendet sie an den dispatcher. Sollte der Inhalt des Puffers die maximale Länge eines Ethernet Frames übersteigen, so wird eine weitere MultiFieldSequenceMessege erstellt, die in einem separaten Ethernet Frame übertragen wird. Durch den TimeTriggeredBuffer erhält der dispatcher die notwendige Information der Übertragungsart über das Backbone-Netzwerk sowie den virtuellen Link bzw. die Mac-Adresse. Mit dieser Information generiert der dispatcher eine InterConnectMsg und gibt sie an das übergeordnete PreBuffer-Modul zurück. Das PreBuffer-Modul beendet den Kreislauf innerhalb der Gateway-Logik, indem es die Nachricht an das InOutComing-Modul weiterleitet.

**Funktion als Destination-Gateway**
Im Destination-Gateway findet die Funktion der Nachrichten-Akkumulation keine Anwendung. Eintreffende Nachrichten werden grundsätzlich umgehend an das nachfolgende Modul InOutComing weitergeleitet.

### 5.2.9 Erweiterungsmöglichkeiten um weitere Bussysteme

Dieses Kapitel beschreibt die Erweiterungsmöglichkeiten des Simulationsmodells um weitere Bussysteme. Im Folgenden wird beschrieben, wie die spezifischen Anforderungen weiterer Bussysteme auf das Simulationsmodell übertragen werden können.

Im Fall einer Erweiterung müssen die Algorithmen im Transformation-Modul ausgeweitet werden. Um den Verwaltungsaufwand während der Simulation möglichst gering zu halten, befinden sich Algorithmen und Logik beide im Transformations-Modul, wobei jeder Algorithmus in einer separaten Funktion definiert ist. Die Logik ruft die entsprechend ausgelagerten Algorithmen auf. Zusätzliche Algorithmen müssen entsprechend in der Logik referenziert werden. Bussystem-spezifische Protokollfelder, die in der FieldSequenceDataStructure (Transport-Darstellung) über das Backbone-Netzwerk übertragen werden sollen, müssen erbend von der Basis-Klasse FieldElement erstellt werden.

Um die Kommunikation mit einem weiteren Bussystem zu ermöglichen, muss ein Teilnehmer dieses Bussystems mit entsprechendem Gateway-Verhalten angeschlossen werden. Dieser muss wie der CANGatewayNode die Verbindung zwischen Bussystem und Gateway gewährleisten. Um die Bus-Frames innerhalb der GatewayBase korrekt zu verarbeiten, muss das Frame-Format in den GatewayBase-Modulen berücksichtigt werden.

Der Puffer für die Nachrichten-Akkumulation wird nach der Priorität der Nachrichten sortiert. Das Prioritätskonzept des entsprechenden Bussystems muss in die Sortierung der Nachrichten integriert werden.

## 5.3 Anwendung des Simulationsmodells

In den folgenden Unterkapiteln wird die Anwendung des Simulationsmodells vorgestellt.

### 5.3.1 Konfiguration

Die gesamte Konfiguration des Gateway-Verhaltens lässt sich über die Routing-Tabelle konfigurieren. Die Konfiguration der Bussysteme und des Time-Triggered Ethernet müssen nach der Spezifikation der jeweiligen Simulationsmodelle zusätzlich mittels ini-Files konfiguriert werden.

Um die Verzögerungszeit durch das Gateway möglichst gering zu halten, muss die Anbindung zwischen GatewayBase und den Applikationen in der Netzwerk-Designphase möglichst eng gestrickt werden. Die Herausforderung liegt darin, die Nachrichten möglichst ohne Zeitverzögerung an das Netzwerk oder Bussystem weiterzuleiten.

Der Sendewunsch des Gateways ist nicht genau vorhersagbar, da sich die Puffer-Leerung der Nachrichten-Akkumulation nach den aktuell enthaltenen Nachrichten dynamisch an die in der Routing-Tabelle spezifizierten Wartezeiten anpasst. Für das TTEthernet Backbone-Netzwerk ist eine Weiterleitung der Nachrichten als TT- oder RC-Nachrichten denkbar. Werden TT-Nachrichten zur Weiterleitung verwendet, so kann es zu einer Verzögerung kommen, wenn Puffer-Leerung der Nachrichten-Akkumulation und Sende-Zyklus der TT-Nachricht nicht direkt aufeinander treffen. Kürzere Sende-Zyklen können bei diesem Problem Abhilfe schaffen, da nur dann eine Nachricht verschickt wird, wenn tatsächlich Daten vorliegen. Die Verwendung von RC-Nachrichten zur Weiterleitung im TTEthernet Backbone-Netzwerk hat den Vorteil, dass RC-Nachrichten event-gesteuert versendet werden. Bei RC-Nachrichten treten Sendeverzögerungen auf, wenn gleichzeitig TT-Nachrichten, die eine höhere Priorität als RC-Nachrichten besitzen, über denselben Netzwerkabschnitt gesendet werden sollen. Die Anbindung an das CAN-Bussystem ist weniger problematisch, da das CAN-Bussystem alle Nachrichten event-gesteuert versendet. Durch die Verwendung von CAN-IDs mit hoher Priorität kann die Sendeverzögerung in Richtung CAN-Bussystem sehr gering gehalten werden.

Lässt die Anwendung keine Verzögerung zu, so kann auf eine Nachrichten-Akkumulation verzichtet werden, indem der Parameter messageAccumulation in der Routing-Tabelle auf false gesetzt wird. Der Sendewunsch des Gateways ist damit bekannt. Durch TT-Nachrichten kann auf diese Weise eine sehr enge Anbindung zwischen Gateway und TTEthernet Backbone Netzwerk konfiguriert werden.

Für eine automatisierte Konfiguration großer Automobil-Netzwerke wurde in der Forschungsgruppe CoRE ein Tool[30] entwickelt, das auf Grundlage einer Spezifikationssprache alle erforderlichen Module und Konfigurationen einer OMNeT++-Simulation generiert.

### 5.3.2 Anwendungsbeispiel

Durch ein Anwendungsbeispiel des Simulationsmodells eines Multi-Bus Realtime Ethernet Gateways soll der Bezug zur Praxis hergestellt werden. In Abbildung XXXV ist ein Automobil-Netzwerk mit Backbone-Architektur dargestellt. Durch 3 Gateways werden 6 CAN-Bussysteme an das Time-Triggered Ethernet Backbone-Netzwerk angebunden.

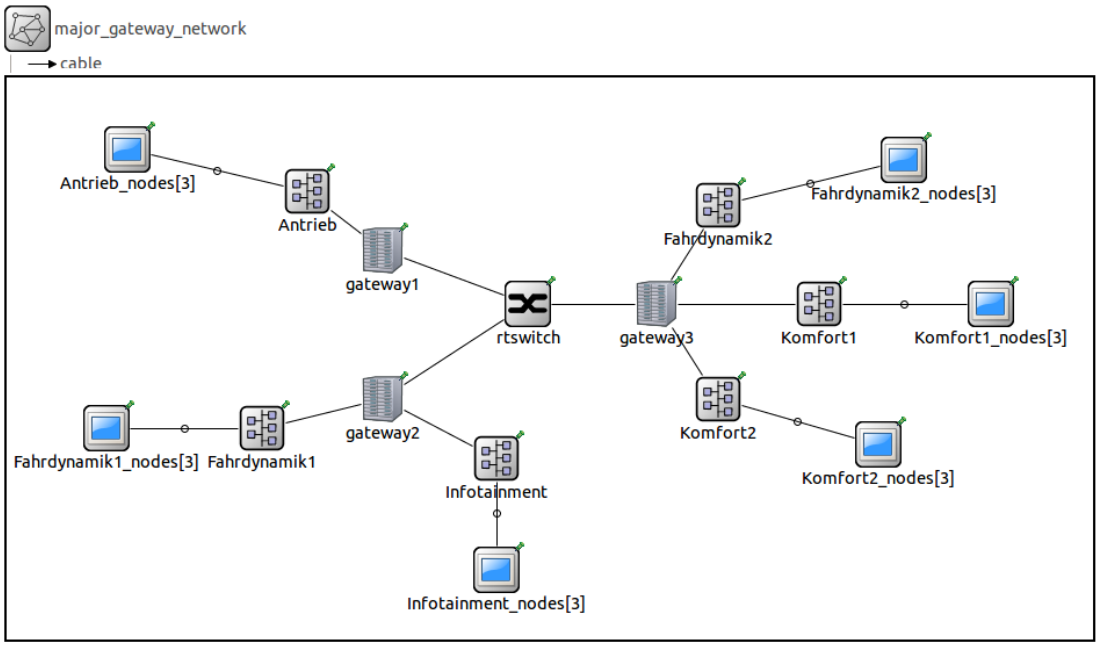


Abbildung XXXV: Simulation eines Autonetzwerkes

Mithilfe der Gateways können Bussysteme aus allen Funktionsbereichen des Automobils miteinander kommunizieren und komplexe, domänenübergreifende Funktionen ausführen.

[30] Siehe (Kempf, 2014)

Ein Ausschnitt der Routing-Tabelle mit der Konfiguration dieses Netzwerkes befindet sich im Anhang. (Siehe Anhang B)

# 6 Verifikation und Test

Dieses Kapitel stellt die Verifikation des Simulationsmodells des Multi-Bus Realtime Ethernet Gateways dar und gibt es einen Einblick in die Software-Tests.

## 6.1 Verifikation

In diesem Teil der Arbeit soll sichergestellt werden, dass das Simulationsmodell auf Funktionsebene gemäß der spezifizierten Anforderungen in Kapitel 3 erstellt wurde.

### 6.1.1 Routing der Nachrichten

Die Routing-Tabelle ist flexibel konfigurierbar, sodass vielfältige Konfigurationsmöglichkeiten realisiert werden können. Die Konfigurierbarkeit ist so umfangreich, dass sie weit über die Grundfunktionalität hinausgeht. Für jede Nachricht kann eine Vielzahl von Zielen definiert werden, unabhängig davon, ob das Ziel ein Bussystem des eigenen Gateways oder eines entfernten Gateways ist. Es können alternative Routingpfade definiert werden, um eine redundante und damit sichere Übertragung von Nachrichten zu ermöglichen.

### 6.1.2 Transformation der Nachrichten

Die Transformationsalgorithmen wurden nur für das CAN-Bussystem gemäß Kapitel 4.4 implementiert. Wie spezifiziert, werden CAN-Frames in eine Transport-Darstellung umgeformt, die es erlaubt, eine weitere Transformation in eine beliebige Nachrichten-Darstellung durchzuführen. Die Möglichkeit, eine Transformation in andere Bussysteme durchzuführen, ist implementiert.

### 6.1.3 Nachrichten-Akkumulation

Die Nachrichten-Akkumulation kann für jede Nachricht in der Routing-Tabelle spezifiziert werden. Durch das messageAccumulation-Flag kann sie für jede Nachricht aktiviert bzw. deaktiviert werden. In der holdUpTime kann für jede Nachricht eine maximale Wartezeit definiert werden. Sie wird mit Zeiteinheit angegeben, sodass die Routing-Tabelle aussagekräftige Information darstellt und verarbeitet.

### 6.1.4 Zeitanforderung

Die Verzögerung im Gateway setzt sich aus den Kriterien Bearbeitung, Verzögerung für Nachrichten-Akkumulation und Sendeverzögerung zusammen. Abgesehen von der Nachrichten-Akkumulation, treten diese Kriterien sowohl im Source-Gateway als auch im Destination-Gateway auf. Angesichts der gesamten Latenzzeit spielt die Übertragungszeit über das Backbone-Netzwerk auch eine Rolle, die aber wegen des geringen Einflusses nicht weiter beachtet werden muss. In der folgenden dargestellten Formel sind die relevanten Kriterien der Verzögerung aufgeführt.

$$t_{GW-S-B} + t_{GW-S-NA} + t_{GW-S-SV} + t_{GW-D-B} + t_{GW-D-SV}$$

Der Index GW-S symbolisiert das Source-Gateway und GW-D symbolisiert das Destination-Gateway. Durch B wird die Bearbeitungszeit indiziert, NA indiziert die Nachrichten-Akkumulation und SV die Sendeverzögerung.

Durch die adaptive Arbeitsweise der Nachrichten-Akkumulation ist eine Zeitmessung der Verzögerung im Gateway eine komplexe Aufgabe. Eine Studie über die Verzögerungszeit eines Gateways mit dynamischer Nachrichten-Akkumulation könnte in einer separaten Projektarbeit durchgeführt werden.

### 6.1.5 Fehlerbehandlung

Tritt ein Fehler während der Simulation auf, so wird eine aussagekräftige Fehlermeldung generiert. Ist der Fehler für die Fortsetzung der Simulation nicht relevant, so wird die entsprechende Fehlermeldung auf der Konsole ausgegeben und die Simulation fortgesetzt. Fehler, die zu einem Fehlverhalten der Simulation führen, werden durch eine Exception propagiert, die eine genaue Beschreibung des Fehlers beinhaltet und die Simulation stoppt. Ein Fortsetzen der Simulation ist in dem Fall nicht möglich.

### 6.1.6 Anforderungen zur Nutzung von CAN-Bussystemen

Das Gateway unterstützt sowohl Daten- als auch Remote-Frames des CAN-Bussystems. Aus einem CAN-Frame werden die Protokollfelder CAN-ID, RTR-Flag und das Daten-Feld weitergeleitet.

Wie in den Anforderungen spezifiziert, werden Error-Frames von CAN-Bussystemen nicht über das Gateway hinaus propagiert. Die Systemsicherheit kann durch die Applikationen der einzelnen CAN-Teilnehmer erfolgen. (Siehe Kapitel 3.1)

### 6.1.7 Erweiterungsmöglichkeiten

Erweiterungsmöglichkeiten sind hier als Möglichkeiten zur Anbindung weiterer Bussysteme definiert. Jedes Bussystem hat spezielle Eigenschaften, die bei einer Erweiterung des

Gateways berücksichtigt werden müssen. Das Simulationsmodell enthält entsprechende Möglichkeiten die in Kapitel 5.2.9 dargestellt sind.

## 6.2 Qualitätssicherung durch Systemtest

Um die Qualität der Software sicherzustellen, wurde ein Systemtest des Simulationsmodells durchgeführt. Anhand eines Beispiel-Netzwerkes sind sämtliche Testfälle ausgeführt worden. Insgesamt konnten mehr als 15 Defects festgestellt und behoben werden. Hier folgt eine Übersicht der wesentlichen Defects.

| Defect | Solution |
|---|---|
| **Ist eine backboneCTID in mehreren Destination-Elementen definiert, wird für jeden Eintrag eine Nachricht erstellt. Es darf nur eine pro backboneCTID erstellt werden.** | Registrierung der backboneCTID an die gesendet wurde in einer Liste und Kontrolle vor dem Senden. |
| **CanDataFrames an interne CAN-Bussysteme gehen immer an die erste Destination.** | Ergänzung einer DestinationCount-Variable, die angibt, welche Destination genutzt werden soll. |
| **Weiterleitung eines CanDataFrames an CAN-Bussystem desselben Gateways führt keine Transformation in die destinationObjectID (CAN-ID) durch.** | Berücksichtigung des Spezialfalles innerhalb der Transformation. |
| **Die Daten aus dem CanDataFrame werden fehlerhaft übertragen.** | Daten-Feld Transformation korrigiert. |
| **InterConnectMsg wird nicht ordnungsgemäß gelöscht.** | Lösch-Operation entsprechend angepasst. |

Abbildung XXXVI: Wesentliche Defects aus dem Systemtest

Nach Behebung der aufgedeckten Defects konnten die Testfälle fehlerfrei ausgeführt wird. In der folgenden Abbildung XXXVII ist eine Übersicht der wichtigsten Testfälle dargestellt.

| Testfall | Iteration | Status |
|---|---|---|
| **Tracking der Daten vom Source-Gateway bis zum Destination-Gateway** | TT,RC,BE | **Passed** |
| **Nachrichtenzuordnung bei mehreren CAN-Bussystemen an einem Gateway** | | **Passed** |
| **Nachricht über mehrere virtuelle Links mit unterschiedlichem Übertragungstyp** | | **Passed** |
| **Nachricht an mehrere Destination-Gateways** | | **Passed** |
| **Funktionstest der Nachrichten-Akkumulation** | | **Passed** |
| **Fehlersimulation** | | **Passed** |

Abbildung XXXVII: Übersicht der wichtigsten Testfälle

Das Testvorgehen wurde mittels eines Debugging-Tools durchgeführt. Durch Break-Points in den einzelnen Modulen konnten Daten und Status während der Simulation verfolgt und analysiert werden.

Die Testfälle decken den größten Teil der Gateway-Funktionen ab, sodass eine fehlerfreie Grundfunktionalität des Gateways gegeben ist.

# 7 Zusammenfassung und Ausblick

Thema dieser Arbeit war die Entwicklung eines Simulationsmodells des Multi-Bus Realtime Ethernet Gateways. Das Konzept des Gateways wurde mit mehreren CAN-Bussystemen und dem Time-Triggered Ethernet als Backbone-Netzwerk implementiert. Die Erweiterung um weitere als auch unterschiedliche Bussysteme ist ohne weiteres möglich.

Einleitend wurden - nach heutigem Stand der Technik - die Grundlagen von Automobil-Netzwerken dargestellt. Angesichts rasanter technologischer Entwicklungen in der Automobil-Industrie wurden die Anforderungen an ein Gateway deutlich, das unterschiedliche Bussysteme über eine Backbone-Architektur zusammenführt. Transformation, Routing und Nachrichten-Akkumulation spielen im Gateway eine wesentliche Rolle. In dieser Arbeit wurden Lösungsansätze entwickelt und implementiert, die ein Spektrum von Funktionen und Konfigurationsmöglichkeiten darstellen. Dies hat im Wesentlichen zu folgenden Ergebnissen geführt: Durch die einheitliche Konfiguration in der Routing-Tabelle kann eine einfache und übersichtliche Konfiguration des Netzwerkes durchgeführt werden. Der dynamische Ansatz der Nachrichten-Akkumulation sorgt für ein adaptives Verhalten des Gateways.

Mit dem Simulationsmodell hat sich die Umsetzung des Gateway-Konzeptes für CAN-Bussysteme bestätigt, dessen Qualität und Funktionalität durch Verifikation und Test sichergestellt werden konnten.

Um das adaptive Verhalten des Multi-Bus Realtime Ethernet Gateways noch besser mit dem Backbone-Netzwerk zu vereinen, könnte IEEE 802.1 Time-Sensitive-Networking (AVB) als zusätzliche Option im Backbone-Netzwerk geringere Latenzzeiten bewirken. Um eine ganzheitliche Abbildung moderner Automobil-Netzwerke in einer Backbone-Architektur zu simulieren, ist die Erweiterung des Gateways um weitere Bussysteme wie FlexRay sinnvoll.

Die Anwendung von Backbone Netzwerk-Architekturen beschränkt sich nicht nur auf die Automobil-Industrie. Auch in der Raum- und Luftfahrt werden Backbone-Architekturen mit Time-Triggered Ethernet diskutiert.

# Literaturverzeichnis

**Association for Standardisation of Automation and Measuring Systems. 2013.** *Data Model for ECU Network Systems (Field Bus Data Exchange Format).* Spezifikation : ASAM e.V., 2013.

**Ayed, Hamdi, Mifdaoui, Ahlem and Fraboul, Christian. 2012.** *Frame Packing Strategy within Gateways for Multi-cluster Avionics Embedded Networks.* Toulouse : s.n., 2012.

**CoRE Research Group.** CoRE. *Communication over Real-Time Ethernet Group.* [Online] HAW Hamburg. http://core.informatik.haw-hamburg.de/en/.

**—.** CoRE Simulation Models for Real-time Networks. [Online] [Cited: 09 2014, 25.] http://core4inet.realmv6.org/.

**Depke, Jan. 2013.** *Spezifikation von Protokolltransformationen für automotive Anwendungen.* Hamburg : s.n., 2013.

**https://www.tttech.com/.** https://www.tttech.com/. [Online] TTTech Computertechnik AG. https://www.tttech.com/.

**Kempf, Fabian. 2014.** *Simulationsbasierte Analyse heterogener Fahrzeugnetzwerke: Generierung, Simulation und Evaluation.* Hamburg : s.n., 2014.

**Lim, Hyung-Taek, Völker, Lars and Herrscher, Daniel. 2011.** *Challenges in a Future IP/Ethernet-based In-Car Network for Real-Time Applications.* New York : IEEE, 2011. 978-1-4503-0636-2 .

**OMNeT++ Community.** OMNeT++. *OMNeT++.* [Online] OMNeT++ Community. www.omnetpp.org.

**OMNeT++ Manual.** OMNeT++ User Manual. [Online] [Cited: 09 23, 2014.] http://www.omnetpp.org/doc/omnetpp/manual/usman.html#toc_3.

**Robert Bosch GmbH. 1991.** CAN Specification. [Online] 1991. [Cited: 09 2014, 26.] http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf.

**SAE International Group. 2011.** *Time-Triggered Ethernet.* International : s.n., 2011. AS6802.

**Simulcraft Inc.** OMNEST High-Performance Simulation for All Kinds of Networks. [Online] Simulcraft Inc. [Cited: 09 23, 2014.] http://www.omnest.com/.

**Steinbach, Till, Korf, Franz and Schmidt, Thomas C. 2011.** *Real-time Ethernet for Automotive Applications: A Solution for Future In-Car Networks.* Hamburg : IEEE, 2011.

**TTTech Computertechnik AG. 2008.** *TTEthernet Specification.* Wien : s.n., 2008. D-INT-S-10-002.

**Wallentowitz, Henning and Reif, Konrad. 2006.** *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen.* Aachen : Friedr. Vieweg & Sohn Verlag, 2006. 978-3-528-03971-4.

**Zimmermann, Prof. Dr.-Ing. Werner and Schmidgall, Dr. Ralf. 2014.** *Bussysteme in der Fahrzeugtechnik.* Stuttgart : Springer Vieweg, 2014. 978-3-658-02418-5.

**—. 2011.** *Bussysteme in der Fahrzeugtechnik.* Wiesbaden : Vieweg + Teubner Verlag | Springer Fachmedien Wiesbaden GmbH, 2011. ISBN: 978-3-8348-0907-0.

# Abbildungsverzeichnis

# Anhang

**Anhang A (Kapitel 5.3)**

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT routingTable            ((item)*)>
<!ATTLIST routingTable   DTDversion     CDATA #REQUIRED >
<!-- DTD version: 1.2 -->

<!ELEMENT item                    ((source), (destination)+, gatewayOptions)>

<!ELEMENT source (sourceBusID, sourceObjectID, sourceType)>
<!ELEMENT sourceBusID             (#PCDATA)>
<!ELEMENT sourceObjectID          (#PCDATA)>
<!ELEMENT sourceType              (#PCDATA)>

<!ELEMENT destination (destinationBusID, destinationObjectID, destinationType,
(backbone)*)>
<!ELEMENT destinationBusID        (#PCDATA)>
<!ELEMENT destinationObjectID     (#PCDATA)>
<!ELEMENT destinationType         (#PCDATA)>

<!ELEMENT backbone ((backboneCTID | directMacAdress), backboneTransferType,
messageAccumulation, holdUpTime)>
<!ELEMENT backboneCTID            (#PCDATA)>
<!ELEMENT directMacAdress         (#PCDATA)>
<!ELEMENT backboneTransferType    (#PCDATA)>
<!ELEMENT messageAccumulation     (#PCDATA)>
<!ELEMENT holdUpTime              (#PCDATA)>

<!ELEMENT gatewayOptions (allwaysSendOldAndNew)>
<!ELEMENT allwaysSendOldAndNew    (#PCDATA)>
```

**Anhang B (Kapitel 5.3.2)**

```
<routingTable>
…
   <item>
      <source>
         <sourceBusID>
            Fahrdynamik1
         </sourceBusID>
         <sourceObjectID>
            7
         </sourceObjectID>
         <sourceType>
            can
         </sourceType>
      </source>
      <destination>
         <destinationBusID>
            Fahrdynamik2
         </destinationBusID>
         <destinationObjectID>
            7
         </destinationObjectID>
         <destinationType>
            can
         </destinationType>
         <backbone>
            <backboneCTID>
               55
            </backboneCTID>
            <backboneTransferType>
               TT
            </backboneTransferType>
            <messageAccumulation>
               true
            </messageAccumulation>
            <holdUpTime>
               8ms
            </holdUpTime>
         </backbone>
      </destination>
      <gatewayOptions>
         <allwaysSendOldAndNew>
```

```
          false
        </allwaysSendOldAndNew>
      </gatewayOptions>
    </item>

    <item>
      <source>
        <sourceBusID>
          Fahrdynamik2
        </sourceBusID>
        <sourceObjectID>
          5
        </sourceObjectID>
        <sourceType>
          can
        </sourceType>
      </source>
      <destination>
        <destinationBusID>
          Fahrdynamik1
        </destinationBusID>
        <destinationObjectID>
          5
        </destinationObjectID>
        <destinationType>
          can
        </destinationType>
        <backbone>
          <backboneCTID>
            56
          </backboneCTID>
          <backboneTransferType>
            TT
          </backboneTransferType>
          <messageAccumulation>
            false
          </messageAccumulation>
          <holdUpTime>
            0ms
          </holdUpTime>
        </backbone>
      </destination>
```

```
      <gatewayOptions>
        <allwaysSendOldAndNew>
          false
        </allwaysSendOldAndNew>
      </gatewayOptions>
    </item>
…
</routingTable>
```

# Versicherung über Selbstständigkeit

*Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

*Hamburg, den* _____ _____